BORLAND® C++

REFERENZ-
HANDBUCH

BORLAND

Borland C++

Borland C++ 3.0

Referenzhandbuch

BORLAND

Copyright © 1991 by Borland International, INC.
Jede Vervielfältigung dieses Handbuches sowie des Softwareprogrammes Borland C++ wird strafrechtlich verfolgt. Die Rechte an der Dokumentation und die Rechte an dem Softwareprogramm Borland C++ liegen bei Borland International.
Der rechtmäßige Erwerb der Programmdisketten und der Handbücher erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daß ein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm Borland C++ nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.
Sie (der Lizenznehmer von Borland C++) dürfen eigene Programme, die Sie mit dem Compiler Borland C++ geschrieben und compiliert haben, ohne Einschränkung oder Verpiflichtung zu zusätzlichen Lizenzgebühren verwenden, weitergeben oder verkaufen, solange diese Programme einen Copyright-Vermerk enthalten. Unter Copyright-Vermerk verstehen wir entweder Ihren eigenen Copyright-Vermerk oder gegebenenfalls einen Hinweis wie »Created using Borland C++ by Borland International 1987, 88, 89, 90, 91«. Die Originaldisketten von Borland C++ beinhalten einige Dateien mit Hardware- und Zeichensatzinformationen, die von der Standard-Grafikbibliothek (GRAPHICS.LIB) verwendet werden. Diese Dateien mit den Suffixen CHR und BGI unterliegen besonderen Nutzungsbeschränkungen. Sie dürfen nicht als Dateien weiterverkauft werden. Sie dürfen diese Dateien aber in Ihren eigenen mit Borland C++ geschriebenen und compilierten Programmen verwenden. Sie dürfen diese compilierten Programme auch weiterverkaufen, solange sie einen Copyright-Vermerk tragen.
Die Beispielprogramme von Borland C++, die auf den Originaldisketten mitgeliefert werden, demonstrieren die verschiedenen Eigenschaften von Borland C++. Sie dienen ausschließlich zu Lehrzwecken. Sie dürfen diese Beispielprogramme editieren oder für den eigenen Gebrauch modifizieren. Sie dürfen diese Programme jedoch nicht als Ganzes oder in Teilen in compilierter Form oder als Quelltext weitergeben oder verkaufen. Sie dürfen die Routinen aus den Beispielprogrammen in eigenen Anwenderprogrammen verwenden, solange die resultierenden Programme nicht einem Beispielprogramm oder einem Teil eines Beispielprogrammes in Erscheinung oder Funktion gleichen.

Einschränkung der Gewährleistung
Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Handbuches übernommen. Da sich Fehler, trotz aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Herausgeber
Borland GmbH
Gautinger Straße 10
Postfach
8130 Starnberg
Telefon 0 18 51 / 2 63 - 0
Telefax 081 51 / 142 09

Übertragung ins Deutsche: Peter Baier, Roland Konstanty, Waltraud Kopelent, Arne Schäpers

Typographische Konzeption des Innenteils: Rudolf Paulus Gorbach

Grafiken: Andreas Höhne, Sid Gastl

Satz: Hanna Bergbauer

PostScript-Belichtung: CD Computer & Dokumentations GmbH, Neuler

Druck und Bindung: Schoder Druck GmbH, Gersthofen

Papier: Schleipen GmbH. Bad Dürkheim. Das verwendete Papier ist säurefrei und alterungsbeständig. Die Produktion erfolgt in einem geschlossenen Wasserkreislauf ohne Einsatz von optischen Aufhellern und unter Verwendung von chlorarm gebleichten, holzfreien Zellstoffqualitäten. Die Abwasserreinigung wird in einer betriebseigenen chemisch-biologischen Kläranlage vorgenommen.

2., unveränderte Auflage, Mai 1992

Inhalt

Einführung 1

Klassen und Klassen-Elementfunktionen 2
Zur Typographie in diesem Handbuch 2

1. Die Funktion main 3

Die Parameter von main 3
Ein Beispielprogramm 4
Jokerzeichen als Kommandozeilenparameter 5
-p verwenden (Pascal-Aufrufkonventionen) 6
Der Rückgabewert von main 6

2. Die Bibliotheksfunktionen 9

Der Aufbau eines Funktionseintrages 9

3. Globale Variablen 633

Index 645

Einführung

Das vorliegende Handbuch enthält die Definitionen der Bibliotheksfunktionen der C++Compiler von Borland, der globalen Variablen und der als global definierten Datentypen zusammen mit Erläuterungen, Hinweisen und Programmbeispielen.

Beachten Sie bitte: Nur der Borland C++Compiler stellt *alle* in diesem Band dokumentierten Funktionen, Variablen und Datentypen zur Verfügung, da auch nur dieser in der Lage ist, sowohl DOS- als auch Windows-Programme zu erzeugen. Besitzen Sie einen der »spezialisierten« Turbo C++Compiler (für DOS oder für Windows), dann ist für diesen Compiler nur ein Subset der hier dokumentierten Funktionen verfügbar. In Zweifelsfällen gibt die kleine Tabelle am Beginn der Funktionsbeschreibung Auskunft darüber, ob die betreffende Funktion für das jeweilige Betriebssystem verfügbar ist.

Dieses Handbuch ist folgendermaßen aufgebaut:

Kapitel 1 »Die Funktion main« beschreibt die Besonderheiten der speziellen Funktion *main*, die in jedem C-Programm enthalten sein muß. Insbesondere wird gezeigt, wie die Parameter, die über die Kommandozeile eingegeben werden, an *main* übergeben werden und wie sie verarbeitet werden können. Außerdem finden Sie Informationen über den Rückgabewert von *main* und die Pascal-Aufrufkonventionen sowie einige Programmbeispiele.

Kapitel 2 »Die Bibliotheksfunktionen« führt sämtliche Funktionen und Makros in alphabetischer Reihenfolge auf. Jede Besprechung besteht aus einer Erläuterung des Namens, einer Syntaxdefinition, einem Verweis auf die dazugehörigen Include-Dateien, einer ausführlichen Besprechung des Zwecks und eventueller Grenzen, einem Hinweis auf die Portabilität, einem Querverweis auf Routinen ähnlicher Funktion und Arbeitsweise sowie einem Programmbeispiel.

Kapitel 3 »Gobale Variablen« definiert und beschreibt die globalen Variablen. Sie können eine Menge Zeit sparen, indem Sie diese Variablen (Datums- und Zeitangaben, Fehlermeldungen, Stackgröße etc.) übernehmen, anstatt sie selbst zu deklarieren.

Klassen und Klassen-Elementfunktionen

Kapitel 2 »Die Bibliotheksfunktionen« enthält bestimmte Klassen und Klassen-Elementfunktionen.

Name	Typ	Name	Typ
abs	Elementfunktion	log	Elementfunktion
acos	Elementfunktion	log10	Elementfunktion
arg	Elementfunktion	norm	Elementfunktion
asin	Elementfunktion	pow	Elementfunktion
atan	Elementfunktion	polar	Elementfunktion
bcd	Klasse	real	Elementfunktion
complex	Klasse	sin	Elementfunktion
conj	Elementfunktion	sinh	Elementfunktion
cos	Elementfunktion	sqrt	Elementfunktion
cosh	Elementfunktion	tan	Elementfunktion
exp	Elementfunktion	tanh	Elementfunktion
imag	Elementfunktion		

Zur Typographie in diesem Handbuch

Zur Verdeutlichung von Zusammenhängen verwenden wir in diesem Handbuch mehrere unterschiedliche Schriftarten:

`Schreibmaschine` steht für Syntaxdefinitionen, Eingaben des Benutzers, Programmlistings und Ausgaben, die von Beispielprogrammen erzeugt werden.

Fettschrift kennzeichnet reservierte Wörter von Borland C++ wie **char**, **switch**, **near**, **cdecl** usw. Innerhalb von Programmlistings sind reservierte Wörter nicht speziell gekennzeichnet.

Kursiv wird für Funktionsnamen (wie *printf*) sowie für Namen von Variablen und Bezeichnern im laufenden Text benutzt. Innerhalb von Programmlistings sind Funktionsnamen nicht speziell gekennzeichnet.

Esc für die Tasten Ihrer Tastatur, wie beispielsweise in »Ein Druck auf Esc entfernt das Menü wieder vom Bildschirm«.

←, ↑ für die Pfeiltasten zur Bewegung des Cursors und andere Spezialtasten.

1. Die Funktion main

Die Funktion *main* muß in jedem C- oder C++-Programm einmal definiert sein. Ihre Position innerhalb des Quelltextes spielt eine untergeordnete Rolle. Während die meisten C-Programmierer *main* dem restlichen Programm voranstellen, werden ehemalige Pascal-Programmierer diese Funktion wohl an das Quelltextende setzen.

Die in den folgenden Abschnitten besprochenen Punkte haben deshalb auch mit der Position von *main* im Quelltext nichts zu tun – sie gelten vielmehr für *jedes* Programm.

Die Parameter von main

main bekommt durch den Startcode des Programms drei Parameter übergeben, die per Konvention als *argc*, *argv* und *env* bezeichnet werden:

- *argc* ist vom Typ **int** und gibt die Anzahl der Kommandozeilenargumente von *argv* an.
- *argv* ist ein Array aus Stringzeigern (**char** *[]):
  - *argv*[*0*] enthält ab der DOS-Version 3.0 den vollen Namen des Programms (inklusive eines eventuellen Suchpfads).
  - *argv*[*0*] zeigt unter den DOS-Versionen vor 3.0 auf den Nullstring ("").
  - *argv*[*1*] zeigt auf den ersten Kommandozeilenparameter.
  - *argv*[2] zeigt auf den zweiten Kommandozeilenparameter.
  - *argv*[*argc-1*] zeigt auf den letzten Parameter, der *main* übergeben wird.
  - *argv*[*argc*] enthält NULL und kennzeichnet so das Ende der Parameterliste.
- *env* ist ebenfalls ein Array aus Stringzeigern. Jedes Element von *env*[] enthält einen Eintrag, der die folgende Form hat: ENVVAR=*Wert*.
  - ENVVAR ist eine *Environment*-Variable, zum Beispiel *PATH* oder *87*.
  - *Wert* ist der Wert, auf den ENVVAR gesetzt ist, zum Beispiel C:\DOS;C:\TOOLS; (für PATH) oder YES (für 87).

Die Namen dieser Parameter sind im Prinzip beliebig (obwohl man sich auch hier an existierende Konventionen halten sollte) – ihre Reihenfolge ist dagegen zwingend vorgeschrieben: *argc*, *argv*, *env*.

Die folgenden Beispiele stellen gültige Deklarationen von *main* dar:

```
main()
main(int argc)              /* zulässig, aber ungewöhnlich */
main(int argc, char *argv[])
main(int argc, char *argv[], char *env[])
```

Die Deklaration *main(int argc)* ist zwar gültig, macht aber nicht viel Sinn – es sei denn, Sie wollen keine Parameter übergeben, sondern nur eine Meldung wie `Kommandozeilenparameter ignoriert` ausgeben lassen.

Der Parameter *env* ist auch über die globale Variable *environ* verfügbar. Auf Einträge der Tabelle *Environment* kann auch dann zugegriffen werden, wenn dieser Parameter nicht definiert ist (siehe die Beschreibung der Funktionen *putenv* und *getenv* in Kapitel 2 »Die Bibliotheksfunktionen« und den Abschnitt über *environ* in Kapitel 3 »Globale Variablen«). Analoges gilt für *argc* und *argv*: das Laufzeitsystem von C++deklariert zwei globale Variablen namens *_argc* und *_argv*, die innerhalb des Programms unabhängig von der Deklaration der Funktion *main* zur Verfügung stehen.

Ein Beispielprogramm

Das folgende Programm trägt den Namen ARGS.EXE und gibt die ihm übergebenen Parameter der Reihe nach aus. Je nachdem, welche Einträge die Tabelle *Environment* Ihres Computers enthält, können sich diese Ausgaben mehr oder weniger von den gezeigten unterscheiden.

```
/* Program ARGS.C */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[], char *env[])
{
   int i;

   printf("The value of argc is %d \n\n",argc);
   printf("These are the %d command-line arguments passed to"
          "main:\n\n",argc);

   for (i = 0; i < argc; i++)
      printf("   argv[%d]: %s\n", i, argv[i]);

   printf("\nThe environment string(s) on this system are:\n\n");

   for (i = 0; env[i] != NULL; i++)
      printf("   env[%d]: %s\n", i, env[i]);
   return 0;
}
```

Starten Sie dieses Programm von der DOS-Bereitschaftsmeldung aus mit der folgenden Zeile:

```
C:> args first_arg "arg with blanks" 3  4 "last but one" stop!
```

Leerzeichen werden als Trenner zwischen einzelnen Parametern behandelt – um einen Parameter wie *argv[2]* zu erzeugen, müssen Sie den entsprechenden String in Anführungszeichen einschließen.

Sie erhalten die folgenden Ausgaben (wobei der vollständige Name von ARGS.EXE und die *Environment*-Einträge von Ihrem Computer abhängig sind):

```
The value of argc is 7

These are the 7 command-line arguments passed to main:

   argv[0]: C:\BC\TESTARGS.EXE
   argv[1]: first_arg
   argv[2]: arg with blanks
   argv[3]: 3
   argv[4]: 4
   argv[5]: last but one
   argv[6]: stop!

The environment string(s) on this system are:

   env[0]: COMSPEC=C:\COMMAND.COM
   env[1]: PROMPT=$p $g
   env[2]: PATH=C:\SPRINT;C:\DOS;C:\BORLANDC
```

Die Maximallänge einer Kommandozeile ist von DOS auf 128 Zeichen begrenzt, wobei Programmname, Suchwege und Trennzeichen zwischen den einzelnen Parametern mitgezählt werden.

Jokerzeichen als Kommandozeilenparameter

Wenn Sie ein C-Programm schreiben wollen, das mehrere Dateien über eine »Suchmaske« (wie z.B. **.C* oder *TEST*.TXT*) bearbeitet, brauchen Sie lediglich die Datei WILDARGS.OBJ zu Ihrem Programm binden. WILDARGS ersetzt die Funktion *setargv* des Startcodes und prüft jeden Kommandozeilenparameter auf die Zeichen »*« bzw. »?«. Wenn diese Zeichen enthalten sind und sich der Parameter zu einem gültigen Dateinamen erweitern läßt, dann folgt ein Absuchen des aktuellen bzw. durch den Parameter angegebenen Verzeichnisses; gefundene Dateinamen werden im Array *argv[]* gespeichert, *argc* wird entsprechend erhöht. Die Maximalgröße des Array *argv* hängt von der Größe des Heap ab, der Ihrem Programm zur Verfügung steht.

Falls sich keine einzige Datei finden läßt, die der Suchmaske entspricht, gibt WILDARGS den Parameter unverändert (d.h. mit den enthaltenen Jokerzeichen) an *main* weiter; in Anführungszeichen eingeschlossene Kommandozeilenparameter verändert bzw. berücksichtigt die Routine grundsätzlich nicht.

Die beiden folgenden Befehle compilieren die Datei ARGS.C, linken sie mit

WILDARGS.OBJ zusammen und starten anschließend die resultierende Datei ARGS.EXE:

```
BCC ARGS WILDARGS.OBJ
ARGS C:\BORLANDC\INCLUDE\*.H "*.C"
```

Bei diesem Aufruf erweitert ARGS.EXE das erste Argument zu allen Namen der *.H-Dateien im INCLUDE-Verzeichnis von Borland C++. Dabei bekommt jeder erweiterte Argumentstring den vollständigen Pfadnamen. Das Argument *.C wird nicht erweitert, weil es in Anführungszeichen steht.

Innerhalb der IDE benötigen Sie für dieses Beispiel eine Projektdatei mit den folgenden beiden Einträgen:

```
ARGS
WILDARGS.OBJ
```

Anschließend müssen Sie noch mit der Option *Run/Arguments* die Kommandozeilenparameter setzen.

Falls Sie WILDARGS grundsätzlich in Ihre Programme aufnehmen wollen, können Sie die Standardfunktion *setargv* mit den folgenden Befehlen in den Bibliotheken ersetzen. Dabei ist unterstellt, daß sich die Bibliotheken und WILDARGS.OBJ im aktuellen Verzeichnis befinden.

```
tlib cs -setargv +wildargs
tlib cc -setargv +wildargs
tlib cm -setargv +wildargs
tlib cl -setargv +wildargs
tlib ch -setargv +wildargs
```

Hinweis: Ausführliche Informationen zu TLIB finden Sie im *Benutzerhandbuch*.

-p verwenden (Pascal-Aufrufkonventionen)

Beim Compilieren eines Programms, das die Pascal-Aufrufkonventionen verwendet, mit dem Compiler-Schalter *-p* (in Kapitel 2 »Lexikalische Grammatik« des *Programmierhandbuches* ausführlich beschrieben) dürfen Sie nicht vergessen, *main* ausdrücklich als C-Routine zu deklarieren. Dies erfolgt mit dem Schlüsselwort **cdecl**:

```
cdecl main(int argc, char * argv[], char * envp[])
```

Der Rückgabewert von main

Solange ein Programm nicht mit *exit* oder *_exit* abgebrochen wird, ist der Rückgabewert von *main* der Statuscode des Programms und vom Typ **int**. Beim

Abbruch eines Programms mit *exit* (oder *_exit*) wird der an *exit* übergebene Wert zurückgegeben. Enthält Ihr Programm zum Beispiel den Aufruf

```
exit(1)
```

so ist der Status 1.

2. Die Bibliotheks-funktionen

In diesem Kapitel werden die Funktionen der C++ Bibliothek der Borland-Compiler ausführlich beschrieben. Einige Funktionen werden als »Funktionsfamilie« gemeinsam erklärt (z.B. die Funktionen der *exec...*- und *spawn...*-Familie), da sie eine ähnliche Arbeitsweise und Aufgabe haben. Jede andere Routine hat ihren eigenen Eintrag und kann unter ihrem Namen gefunden werden. Werden z.B. die Erläuterungen zur Funktion *free* benötigt, muß in dem entsprechenden Abschnitt *free* nachgeschlagen werden. In diesem Eintrag findet man dann:

- eine Kurzbeschreibung von *free*;
- die Syntaxdefinition für den Aufruf von *free*;
- der Name der Include-Datei(en), die den Prototyp von *free* enthalten;
- eine ausführliche Beschreibung der Arbeitsweise von *free* und der Unterschiede zu anderen Speicherverwaltungsroutinen;
- Informationen zur Portabilität bzw. Verwendung mit anderen Compilern mit entsprechenden Funktionen;
- einen Verweis auf verwandte Funktionen;
- und ein Beispielprogramm für die Verwendung der Funktion.

Die Programmbeispiele sind auch im Hilfesystem enthalten und können von dort in Ihre eigenen Programme kopiert werden.

Folgendes Beispiel steht stellvertretend für die Struktur eines Funktionseintrages in diesem Handbuch:

Der Aufbau eines Funktionseintrages

<Beispiel>

Funktion	Eine kurze Beschreibung der Funktion.
Syntax	`#include <header.h>`

Name der Include-Datei(en), die den Prototyp der beschriebenen Funktion sowie Definitionen von Konstanten, **enum**-Datentypen usw. enthält.

```
Ergebnistyp Funktionsname(Parameter, Parameter[,...]);
```

Die hier dargestellte Syntax entspricht dem Prototyp der Funktion. Die Klammer zeigt an, daß noch weitere Parameter folgen können.

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	x

Die Portabilität der Funktion wird mit einem X in der jeweiligen Spalte des Systems bzw. der Programmiersprache angezeigt. Falls es Einschränkungen gibt, sind sie im Beschreibungsabschnitt aufgeführt.

DOS — Auf DOS-Systemen verfügbar.

UNIX — Auf UNIX-Systemen verfügbar.

Windows — Kompatibel zu Windows. Wenn Sie mit EasyWin arbeiten, lesen Sie bitte in Anhang C »EasyWin« des *Benutzerhandbuchs* nach. Dort wird unter anderem erläutert, wie nicht zu Windows kompatible Funktionen (zum Beispiel *printf* und *scanf*) in Programmen eingesetzt werden können, die unter Windows ausgeführt werden.

ANSI C — In ANSI C definiert.

nur C++ — Erfordert C++. Nicht in ANSI C definiert.

Sind in einem Eintrag mehrere Funktionen beschrieben, wird die Portabilität dieser Funktionen in einer mehrzeiligen Tabelle dargestellt.

Beschreibung — Zweck und Verwendung der aufgeführten Funktionen, Beschreibung der benutzen Parameter und weitere Details.

Rückgabewert — Typ und Bereich des Rückgabewerts (soweit vorhanden). Wenn die Funktion globale Variablen setzt, finden Sie hier eine Aufstellung der entsprechenden Werte.

Siehe auch — Auflistung anderer Routinen, die mit der beschriebenen Funktion in irgendeinem Zusammenhang stehen. Bei Querverweisen wie *...Funktionsname* oder *Funktionsname...* wird auf ganze Funktionsfamilien hingewiesen (*exec...* bezieht sich also auf alle *exec*-Funktionen wie *execl*, *execlp*, *execlpe*, *execlv*, *execve*, *execvp* und *execvpe*)

Beispiel — Programmlisting, in dem die Routine verwendet wird. Alle Programmbeispiele in diesem Kapitel sind auch im Hilfe-

system enthalten. Sie können sie von dort kopieren und in Ihre eigenen Dateien einfügen.

abort

Funktion: Beendet ein Programm mit einer Fehlermeldung.

Syntax:

```
#include <stdlib.h>
void abort(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung: *abort* gibt die Meldung

```
Abnormal program termination
```

über *stderr* aus und bricht das Programm über einen Aufruf von *_exit* ab. Der durch *abort* zurückgelieferte Exit-Code hat den Wert 3.

Rückgabewert: Gibt den Exit-Code 3 an den übergeordneten Prozeß oder an DOS aus.

Siehe auch: *assert*, *atexit*, *exit*, *_exit*, *raise*, *signal*, *spawn...*

Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  printf("Calling abort()\n");
  abort();
  return 0; /* This is never reached */
}
```

abs

Funktion: Liefert den absoluten Betrag einer Integer-Zahl.

Syntax:

reelle Version:
```
#include <math.h>
int abs(int x);
```

komplexe Version:
```
#include <complex.h>
double abs (complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*reelles abs*
x		x		x	*komplexes abs*

Beschreibung: *abs* liefert den absoluten Betrag seines Integer-Arguments *x*. Wenn stdlib.h aufgenommen wurde, wird diese Funktion als

Makro behandelt – ansonsten (bzw. nach der Anweisung #*undef abs*) handelt es sich um einen »echten« Funktionsaufruf.

Rückgabewert Die reelle Version von *abs* liefert einen **int** im Bereich von 0 bis 32767. Der Wert -32768 hat für den Datentyp **int** keine positive Entsprechung, *abs*(*-32768*) liefert deshalb den Wert -32768 zurück.

Die komplexe Version von *abs* liefert ein Ergebnis vom Typ **double**.

Siehe auch *cabs*, *complex*, *fabs*, *labs*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
  int number = -1234;

  printf("number: %d  absolute value: %d\n", number,
         abs(number));
  return 0;
}
```

absread

Funktion Liest einzelne Diskettensektoren.

Syntax

```
#include <dos.h>
int absread(int drive, int nsects, long lsect,
            void *buffer);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Diese Funktion ignoriert die logische Struktur einer Diskette (d.h. die Unterteilung in FAT, Verzeichnisse, Dateien etc.) und bearbeitet einzelne Sektoren direkt.

absread verwendet den DOS-Interrupt 0x25 zum Lesen von Sektoren.

drive ist die Laufwerksnummer (A =0, B =1, C =2, ...), *nsects* die Anzahl der Sektoren, die gelesen werden sollen, *lsect* die Nummer des ersten zu lesenden Sektors und *buffer* der Zeiger auf einen Speicherbereich, in den die Daten gelesen werden sollen.

Mit einem Aufruf von *absread* können maximal 64 KByte gelesen werden. Ist der Puffer kleiner, bestimmt er die Obergrenze.

Rückgabewert Bei fehlerfreier Ausführung liefert *absread* den Wert 0 zurück. Im Fall eines Fehlers wird die globale Variable *errno* auf den vom System gelieferten Wert des Registers AX gesetzt. *absread* hat dann den Rückgabewert -1.

Siehe auch *abswrite, biosdisk*

Beispiel

```
/* absread example */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dos.h>

int main(void)
{
  int i, strt, ch_out, sector;
  char buf[512];

  printf("Insert a diskette into drive A "
         "and press any key\n");
  getch();
  sector = 0;
  if (absread(0, 1, sector, &buf) != 0)
  {
     perror("Disk problem");
     exit(1);
  }
  printf("Read OK\n");
  strt = 3;
  for (i=0; i<80; i++)
  {
     ch_out = buf[strt+i];
     putchar(ch_out);
  }
  printf("\n");
  return(0);
}
```

abswrite

Funktion Schreibt einzelne Diskettensektoren.

Syntax

```
#include <dos.h>
int abswrite(int drive, int nsects, long lsect,
             void *buffer);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Diese Funktion ignoriert die logische Struktur einer Diskette (d.h. die Unterteilung in FAT, Verzeichnisse, Dateien etc.) und bearbeitet einzelne Sektoren direkt.

Hinweis: abswrite kann Dateien, Verzeichnisse und FATs überschreiben, wenn Sie die Parameter entsprechend angeben.

abswrite schreibt Sektoren über den DOS-Interrupt 0x26.

drive ist die Laufwerksnummer (A=0, B=1, C=2, ...), *nsects* die Anzahl der Sektoren, die geschrieben werden sollen, *lsect* die Nummer des ersten zu schreibenden Sektors und *buffer* der Zeiger auf einen Speicherbereich, dessen Inhalt geschrieben werden soll.

Mit einem Aufruf von *abswrite* können maximal 64 KByte geschrieben werden. Ist der Puffer kleiner, bestimmt er die Obergrenze.

Rückgabewert Bei fehlerfreier Ausführung liefert *abswrite* den Wert 0 zurück. Im Fall eines Fehlers wird die globale Variable *errno* auf den vom System gelieferten Wert des Registers AX gesetzt. *abswrite* hat dann den Rückgabewert -1.

Siehe auch *absread*, *biosdisk*

access

Funktion Ermittelt die Zugriffsmöglichkeiten auf eine Datei.

Syntax
```
#include <io.h>
int access(const char *filename, int amode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *access* prüft, ob eine Datei des (als *filename*) angegebenen Namens existiert. Wenn ja, folgt eine Prüfung, ob der in *amode* angegebene Zugriff möglich ist. Die folgenden Werte sind für *amode* definiert:

06 Prüfung auf »Lesen und Schreiben erlaubt«
04 Prüfung auf »Lesen erlaubt«
02 Prüfung auf »Schreiben erlaubt«
01 Prüfung auf »Ausführung erlaubt«
00 Prüfung auf »Datei existiert«

Unter DOS können existierende Dateien weder gegen »Lesen« noch gegen »Ausführen« gesperrt werden – die Werte 04 und 00 für *amode* erzeugen also dasselbe Ergebnis. Eine Schreiberlaubnis impliziert bei DOS auch die Erlaubnis zum Lesen – folglich sind die Werte 06 und 02 für *amode* ebenfalls äquivalent.

Wenn *filename* ein Verzeichnis bezeichnet, prüft *access* nur das Vorhandensein dieses Verzeichnisses.

Rückgabewert Wenn die Datei existiert und der durch *amode* angegebene Zugriff erlaubt ist, liefert *access* den Wert 0 zurück. Ansonsten ist der Rückgabewert -1 und die globale Variable *errno* bekommt einen der folgenden Werte:

ENOENT	Suchweg oder Datei nicht vorhanden
EACCES	Datei vorhanden, aber Zugriff nicht erlaubt

Siehe auch *chmod*, *fstat*, *stat*,

Beispiel

```
#include <stdio.h>
#include <io.h>

int file_exists(char *filename);

int main(void)
{
  printf("Does NOTEXIST.FIL exist: %s\n",
         file_exists("NOTEXISTS.FIL") ? "YES" : "NO");
  return 0;
}

int file_exists(char *filename)
{
  return (access(filename, 0) == 0);
}
```

Programmausgabe:

```
Does NOTEXIST.FIL exist?  NO
```

acos, acosl

Funktion Berechnet den Arcuscosinus.

Syntax

Reelle Versionen:

```
#include <math.h>
double acos(double x);
long double acosl(long double x);
```

Komplexe Versionen:

```
#include <complex.h>
complex acos (complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*acosl*
x	x	x	x		*reelles acos*
x		x		x	*komplexes acos*

Beschreibung *acos* ist die Umkehrfunktion zu *cos*, erwartet einen Wert im Bereich von -1 bis +1 und liefert den dazugehörigen Winkel in der Einheit *rad* zurück. *acosl* ist die **long double** Version und

erwartet ein Argument vom Typ **long double** . Sie liefert ein Ergebnis vom Typ **long double** zurück. *acos* und *acosl* erwarten ein reelles Argument im Bereich von -1 bis +1. Argumente außerhalb dieses Bereichs liefern NAN (»Not a Number«) und setzen *errno* auf den Wert.

EDOM Argument außerhalb des zulässigen Bereichs

Die komplexe Version von *acos* ist wie folgt definiert:

$acos(z) = -i\ log(z + i\ sqrt(1 - z^2))$

Rückgabewert Bei einem reellen Argument zwischen -1 und +1 liefern *acos* und *acosl* Werte im Bereich von *0* bis *pi* zurück. Die Fehlerbearbeitung kann mit *matherr* bzw. *matherrl* modifiziert werden.

Siehe auch *asin*, *atan*, *atan2*, *complex*, *cos*, *matherr*, *sin*, *tan*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
  double result;
  double x = 0.5;

  result = acos(x);
  printf("The arc cosine of %lf is %lf\n", x, result);
  return 0;
}
```

alloca

Funktion Belegt temporären Speicher auf dem Stack.

Syntax

```
#include <malloc.h>
void *alloca(size_t size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *alloca* reserviert einen Speicherbereich von *size* Bytes auf dem Stack. Der reservierte Speicherbereich wird beim Beenden der aufrufenden Funktion freigegeben.

alloca modifiziert den Stack-Zeiger und darf deshalb nicht in einem Ausdruck aufgerufen werden, der ein Argument einer Funktion ist.

Die aufrufende Funktion muß Referenzen auf lokale Variablen im Stack enthalten, da der Stack andernfalls beim Funktionsende nicht korrekt wiederhergestellt werden kann. Ver-

wenden Sie deshalb folgenden Programmtext in der aufrufenden Funktion:

```
char *p;
char dummy[1];

dummy[0] = 0;
...
p = alloca(nbytes);
```

Rückgabewert Wenn im Stack ein Speicherbereich von ausreichender Größe zur Verfügung steht, liefert *alloca* einen Zeiger auf den belegten Speicherbereich zurück. Andernfalls ist der Rückgabewert NULL.

Siehe auch *malloc*

Beispiel

```
#include <stdio.h>
#include <malloc.h>

void test(int a)
{
  char *newstack;
  int len = a;
  char dummy[1];

  dummy[0] = 0;          /* force good stack frame */
  printf("SP before calling alloca(0x%X) = 0x%X\n",
         len,_SP);
  newstack = (char *) alloca(len);
  printf("SP after calling alloca = 0x%X\n",_SP);
  if (newstack)
    printf("Alloca(0x%X) returned %p\n",len,newstack);
  else
    printf("Alloca(0x%X) failed\n",len);
}

void main()
{
  test(256);
  test(16384);
}
```

allocmem, _dos_allocmem

Funktion Reserviert DOS-Speichersegmente für ein Programm.

Syntax

```
#include <dos.h>
int allocmem(unsigned size, unsigned *segp);
int _dos_allocmem(unsigned size, unsigned *segp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

allocmem und *_dos_allocmem* verwenden die DOS-Funktion 0x48, um einen (zusätzlichen) Speicherblock für das aktuelle Programm zu reservieren und liefern die Segmentadresse des belegten Blocks zurück.

Über den Parameter *size* wird die Größe des zu belegenden Blocks in Paragraphen (d.h. Abschnitten zu jeweils 16 Byte) angegeben. *segp* zeigt auf das WORD, dem die Segmentadresse des belegten Blocks zugewiesen wurde.

Ist kein Block der angegebenen Größe verfügbar, führt *allocmem* keine Zuweisung an das WORD durch, auf das *segp* zeigt. *_dos_allocmem* speichert in diesem Fall die Größe (in Paragraphen) des größten verfügbaren Blocks in dem WORD, auf das *segp* zeigt.

Die durch *allocmem* und *_dos_allocmem* belegten Speicherbereiche sind paragraphenweise ausgerichtet.

Hinweis: allocmem und *_dos_allocmem* können nicht mit *malloc* coexistieren.

Rückgabewert

allocmem liefert den Wert -1, wenn ein Speicherbereich der gewünschten Größe reserviert werden konnte; ansonsten wird die Größe des größten noch verfügbaren Speicherblocks (in Paragraphen) zurückgeliefert.

_dos_allocmem liefert den Wert 0, wenn ein Speicherbereich der gewünschten Größe reserviert werden konnte. Andernfalls wird der DOS-Fehlercode zurückgeliefert und das WORD, auf das *segp* zeigt, auf die Größe des größten noch verfügbaren Speicherblocks (in Paragraphen) gesetzt.

Ist *allocmem* bzw. *_dos_allocmem* nicht erfolgreich, wird die globale Variable *_doserrno* gesetzt und die globale Variable *errno* erhält den folgenden Wert:

ENOMEM Nicht genug Platz im Hauptspeicher

Siehe auch

coreleft, *freemem*, *malloc*, *setblock*

Beispiel

```
#include <dos.h>
#include <alloc.h>
#include <stdio.h>

int main(void)
{
  unsigned int size, segp;
  int stat;

  size = 64;              /* (64 x 16) = 1024 bytes */
  stat = allocmem(size, &segp);
```

```
  if (stat == -1)
    printf("Allocated memory at segment: %x\n", segp);
  else
    printf("Failed: maximum number of paragraphs "
           "available is %u\n", stat);

  return 0;
}
```

arc

Funktion

Zeichnet einen Kreisbogen.

Syntax

```
#include <graphics.h>
void far arc(int x, int y, int stangle,
             int endangle, int radius);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

arc zeichnet in der aktuellen Zeichenfarbe einen Kreisbogen mit dem Mittelpunkt (*x*,*y*), wobei *stangle* den Startpunkt und *endangle* den Endpunkt des mit dem Radius *radius* gezogenen Bogens festlegen. Wird für *stangle* der Wert 0 und für *endangle* der Wert 360 angegeben, zeichnet *arc* einen Kreis.

Die Angabe beider Winkel erfolgt in Grad, wobei entgegen dem Uhrzeigersinn gezählt wird (0 Grad = horizontal rechts des Mittelpunktes, 90 Grad = senkrecht über dem Mittelpunkt usw).

Hinweis: Der Parameter *linestyle* für die Linienart wirkt sich auf *arc*, *circle*, *ellipse*, *fillellipse*, *sector* und *pieslice* nicht aus; der Parameter *thickness* für die Liniendicke wird dagegen verwendet.

Achtung: Bei Verwendung eines CGA-Systems im hochauflösenden Modus oder eines Monochrom-Grafikadapters kann es vorkommen, daß die Beispielprogramme für die Verwendung der Grafikroutinen nicht die erwarteten Ergebnisse erzeugen. In diesen Fällen sollte Funktionen, die die Füll- oder Zeichenfarbe ändern (wie *setcolor*, *setfillstyle* und *setlinestyle*) der Wert 1 anstelle einer der symbolischen Farbkonstanten (diese sind in graphics.h definiert) übergeben werden.

Rückgabewert

Keiner.

Siehe auch

circle, *ellipse*, *fillellipse*, *getarccoords*, *getaspectratio*, *graphresult*, *pieslice*, *sector*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 45, endangle = 135;
   int radius = 100;

   /* initialize graphics and local
   variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();

      exit(1);    /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* draw arc */
   arc(midx, midy, stangle, endangle, radius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

arg

Funktion Liefert den Winkel einer Zahl in der komplexen Zahlenebene.

Syntax
```
#include <complex.h>
double arg(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung *arg* liefert den Winkel (in *rad*) der Zahl in der komplexen Zahlenebene.

Die positive reelle Achse hat den Winkel *0*, die positive imaginäre Achse den Winkel *pi/2*. Ist das an *arg* übergebene Argument 0, so ist der Rückgabewert 0.

Rückgabewert *arg*(*x*) liefert *atan2*(*imag*(*x*), *real*(*x*)) zurück.

Siehe auch *complex*, *norm*, *polar*

Beispiel

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary part = " << imag(z)
        << "\n";
   cout << "z has complex conjugate = " << conj(z)
        << "\n";

   double mag = sqrt(norm(z));
   double ang = arg(z);
   cout << "The polar form of z is:\n";
   cout << "   magnitude = " << mag << "\n";
   cout << "   angle (in radians) = " << ang << "\n";
   cout << "Reconstructing z from its polar form "
           "gives:\n";
   cout << "   z = " << polar(mag,ang) << "\n";
   return 0;
}
```

asctime

Funktion Konvertiert Datum/Uhrzeit in einen ASCII-String.

Syntax

```
#include <time.h>
char *asctime(const struct tm *tblock);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *asctime* konvertiert eine Zeitangabe, die als Struktur des Typs *tm* in **tblock* gespeichert ist, in einen String mit 26 Zeichen, der wie *ctime* das folgende Format hat:

```
Mon Nov 21 11:31:54 1983\n\0
```

Rückgabewert *asctime* liefert einen Zeiger auf einen String zurück, der Zeit und Datum enthält. Der String ist eine statische Variable, die mit jedem Aufruf dieser Funktion überschrieben wird.

Siehe auch *ctime*, *difftime*, *ftime*, *gmtime*, *localtime*, *mktime*, *strftime*, *stime*, *time*, *tzset*

Beispiel

```
#include <stdio.h>
#include <string.h>
#include <time.h>

int main(void)       /* sample loading of tm structure  */
{
   struct tm t;
   char str[80];

   t.tm_sec    = 1;  /* Seconds */
   t.tm_min    = 30; /* Minutes */
   t.tm_hour   = 9;  /* Hour */
   t.tm_mday   = 22; /* Day of the Month  */
   t.tm_mon    = 11; /* Month */
   t.tm_year   = 56; /* Year CR    does not */
                     /* include century */
   t.tm_wday   = 4;  /* Day of the week  */
   t.tm_yday   = 0;  /* Does not show in asctime */
   t.tm_isdst  = 0;  /* Is Daylight SavTime; */
                     /* does not show in asctime */

   /* converts structure to null terminated string */

   strcpy(str, asctime(&t));
   printf("%s\n", str);

   return 0;
}
```

asin, asinl

Funktion Berechnet den Arcussinus.

Syntax

reelle Version:

```
#include <math.h>
double asin(double x);
long double asinl(long double x);
```

komplexe Version:

```
#include <complex.h>
complex asin(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*ansinl*
x	x	x	x		*reelles asin*
x		x		x	*komplexes asin*

Beschreibung *asin* liefert für ein reelles Argument den Arcussinus des eingegebenen Werts zurück.

asinl ist die **long double** Version und erwartet ein Argument vom Typ **long double** . *asinl* liefert ein Ergebnis vom Typ **long double** zurück.

asin und *asinl* erwarten ein reelles Argument im Bereich von -1 bis +1. Reelle Argumente außerhalb dieses Bereichs liefern NAN (Not a Number) und setzen *errno* auf den Wert

EDOM Argument außerhalb des zulässigen Bereichs

Die komplexe Version von *asin* ist wie folgt definiert:

$asin(z) = -i * log(i * z + sqrt(1 - z^2))$

Rückgabewert Der Rückgabewert von *asin* und *asinl* liegt für reelle Argumente im Bereich von *-pi/2* bis *pi/2*. Die Fehlerbearbeitung kann für diese Funktionen mit *matherr* und *_matherrl* modifiziert werden.

Siehe auch *acos*, *atan*, *atan2*, *complex*, *cos*, *matherr*, *sin*, *tan*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;

   result = asin(x);
   printf("The arc sin of %lf is %lf\n", x, result);
   return(0);
}
```

assert

Funktion Prüft eine Bedingung und bricht das Programm ab, wenn sie nicht zutrifft.

Syntax

```
#include <assert.h>
void assert(int test);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *assert* ist ein Makro, das bei der Compilierung zu einer **if**-Anweisung erweitert wird.

Wenn *test* zur Laufzeit des Programms den Wert Null (d.h. *false*) ergibt, wird die Meldung

```
Assertion failed:  test, file <Dateiname>
                       line <Zeilennummer>
```

auf *stderr* ausgegeben. Danach erfolgt der Abbruch des Programms mit *abort*.

In dieser Meldung gibt Dateiname den Namen der Quelltextdatei und Zeilennummer die Nummer der Zeile an, in der sich der Aufruf von *assert* befindet.

Wenn das Symbol *NDEBUG vor* der Aufnahme von assert.h im Quelltext (oder mit #*define NDEBUG*) definiert ist, entfernt der Präprozessor sämtliche Aufrufe von *assert*.

Rückgabewert Keiner.

Siehe auch *abort*

Beispiel

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

struct ITEM {
   int key;
   int value;
};

/* add item to list, make sure list is not  null */
void additem(struct ITEM *itemptr) {
   assert(itemptr != NULL);
   /* add item to list */
}

int main(void)
{
   additem(NULL);
   return 0;
}
```

atan, atanl

Funktion Berechnet den Arcustangens.

Syntax

reelle Version:

```
#include <math.h>
double atan(double x);
long double atanl(long double x);
```

komplexe Version:

```
#include <complex.h>
complex atan(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*atanl*
x	x	x	x		*relles atan*
x		x		x	*komplexes atan*

Beschreibung *atan* berechnet den Arcustangens des übergebenen Arguments.

atanl ist die **long double** Version und erwartet ein Argument vom Typ **long double** . *atanl* liefert ein Ergebnis vom Typ **long double** zurück.

Die komplexe Version von *atan* ist wie folgt definiert:

$atan(z) = -0.5\ i\ log((1 + i\ z)\ /\ (1 - i\ z))$

Rückgabewert Der Rückgabewert von *atan* und *atanl* liegt für reelle Argumente im Bereich von *-pi/2* bis *pi/2*. Die Fehlerbehandlung kann für diese Funktionen mit *matherr* und *_matherrl* modifiziert werden.

Siehe auch *acos*, *asin*, *atan2*, *complex*, *cos*, *matherr*, *sin*, *tan*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;

   result = atan(x);
   printf("The arc tangent of %lf is %lf\n",
          x, result);
   return(0);
}
```

atan2, atan2l

Funktion Berechnet den Arcustangens von *y/x*.

Syntax

```
#include <math.h>
double atan2(double y, double x);
long double atan2l(long double y, long double x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*atan2*
x		x			*atan2l*

Beschreibung *atan2* liefert den Arcustangens von *y/x* und erzeugt auch dann noch korrekte Resultate, wenn sich der ergebende Winkel *pi/2* oder *-pi/2* nähert (den Polstellen der Tangensfunktion, für Werte von *x* nahe Null).

Wenn beide Argumente *x* und *y* Null sind, setzt die Funktion *errno* auf den Wert EDOM.

atan2l ist die **long double** Version und erwartet ein Argument vom Typ **long double** . *atanl* liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert *atan2* und *atan2l* liefern einen Wert im Bereich von *-pi* bis *pi*. Die Fehlerbehandlung kann für die Funktionen mit *matherr* und *_matherrl* modifiziert werden.

Siehe auch *acos*, *asin*, *atan*, *cos*, *matherr*, *sin*, *tan*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 90.0, y = 45.0;

   result = atan2(y, x);
   printf("The arc tangent ratio of %lf is %lf\n",
          (y / x), result);
   return 0;
}
```

atexit

Funktion Registriert Routinen, die am Ende eines Programms aufgerufen werden sollen.

Syntax

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung Über *atexit* registrierte Funktionen werden nach dem normalen Ende eines Programms in der Form (**func*)() direkt vor der Rückkehr zum Betriebssystem aufgerufen.

Jeder Aufruf von *atexit* registriert eine weitere Funktion. Der Aufruf registrierter Funktionen geschieht in der durch die Aufrufe von *atexit* festgelegten Reihenfolge nach dem Prinzip »last in – first out« – die als letzte mit *atexit* festgelegte Funktion wird also zuerst ausgeführt. Maximal 32 Funktionen können so registriert werden.

Rückgabewert *atexit* liefert den Wert 0 zurück, wenn die angegebene Funktion aufgenommen werden konnte; falls bereits 32 Funktionen

registriert sind (und damit kein weiterer Platz mehr zur Verfügung steht), ist das Ergebnis ungleich Null.

Siehe auch *abort*, *_exit*, *exit*, *spawn...*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>

void exit_fn1(void)
{
   printf("Exit function #1 called\n");
}

void exit_fn2(void)
{
   printf("Exit function #2 called\n");
}

int main(void)
{
   /* post exit function #1 */
   atexit(exit_fn1);
   /* post exit function #2 */
   atexit(exit_fn2);
   return 0;
}
```

atof, _atold

Funktion Konvertiert einen ASCII-String in eine Gleitkommazahl.

Syntax

```
#include <math.h>
double atof(const char *s);
long double _atold(const char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*atof*
x		x			*_atold*

Beschreibung *atof* konvertiert den durch *s* bezeichneten String in eine Gleitkommazahl und liefert das Ergebnis als **double** zurück.

Der String kann die folgenden Elemente enthalten:

- eine optionale Folge von Leerzeichen und/oder Tabulatoren;
- ein optionales Vorzeichen;
- eine Folge von Ziffern und einen optionalen Dezimalpunkt (die Ziffern können vor oder hinter dem Dezimalpunkt stehen);

- einen optionalen Exponenten *e* oder *E*, gefolgt von einem optional vorzeichenbehafteten Integerwert.

Der String muß dem folgenden Format entsprechen:

[*whitespace*] [*sign*] [*ddd*] [.] [*ddd*] [*e*|*E*[*sign*]*ddd*]

atof interpretiert zusätzlich die beiden Zeichenfolgen *+INF* und *-INF* als positive bzw. negative Unendlichkeit sowie *+NAN* und *-NAN* (für »Not a Number« – ein nicht numerisch darstellbarer Wert).

Die Konvertierung bricht bei dem ersten nicht interpretierbaren Zeichen ab.

_atold ist die **long double** Version und konvertiert den durch *s* bezeichneten String in einen Wert vom Typ **long double**.

strtod und *_strtold* arbeiten ähnlich wie *atof* und *_atold*, erlauben aber eine bessere Fehlerbehandlung und werden daher in manchen Anwendungen bevorzugt.

Rückgabewert *atof* und *_atold* liefern den konvertierten Wert des Eingabestrings zurück. Im Fall eines Überlaufs liefert *atof* (oder *_atold*) den Wert plus oder minus HUGE_VAL (bzw. _LHUGE_VAL) zurück, *errno* wird auf ERANGE gesetzt. Die Routine *matherr* (bzw. *_matherrl*) wird nicht aufgerufen.

Siehe auch *atoi*, *atol*, *ecvt*, *fcvt*, *gcvt*, *scanf*, *strtod*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   float f;
   char *str = "12345.67";

   f = atof(str);
   printf("string = %   s float = %f\n", str, f);
   return 0;
}
```

atoi

Funktion Konvertiert einen ASCII-String in eine Integerzahl.

Syntax

```
#include <stdlib.h>
int atoi(const char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *atoi* konvertiert den durch *s* bezeichneten String in einen **int**. Der String kann folgende Elemente (in der aufgeführten Reihenfolge) enthalten:

- optionale führende Leerzeichen und/oder Tabulatoren;
- ein optionales Vorzeichen;
- eine Ziffernfolge.

Der String muß also folgendem Format entsprechen:

[*whitespace*] [*sign*] [*ddd*]

Die Konvertierung endet beim ersten nicht interpretierbaren Zeichen, eine Prüfung auf eventuelle Überläufe findet nicht statt.

Rückgabewert *atoi* liefert das Ergebnis der Konvertierung oder 0, wenn der zu interpretierende String nicht zu einem Integerwert konvertierbar ist.

Siehe auch *atof, atol, ecvt, fcvt, gcvt, scanf, strtod*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   int n;
   char *str = "12345.67";

   n = atoi(str);
   printf("string = %s integer = %d\n", str, n);
   return 0;
}
```

atol

Funktion Konvertiert einen ASCII-String in einen **long-Wert**

Syntax

```
#include <stdlib.h>.h
long atol(const char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *atol* konvertiert den durch *s* bezeichneten String in einen **long**-Wert. Der String kann folgende Elemente (in der aufgeführten Reihenfolge) enthalten:

- optionale führende Leerzeichen und/oder Tabulatoren;
- ein (optionales) Vorzeichen;

- eine Ziffernfolge.

Der String muß also folgendem Format entsprechen:

[*whitespace*] [*sign*] [*ddd*]

Die Konvertierung endet beim ersten nicht interpretierbaren Zeichen. Eine Prüfung auf eventuelle Überläufe findet nicht statt.

Rückgabewert *atol* liefert das Ergebnis der Konvertierung zurück. Wenn der zu interpretierende String nicht zu einem **long**-Wert konvertierbar ist, liefert *atol* 0 zurück.

Siehe auch *atof*, *atoi*, *ecvt*, *fcvt*, *gcvt*, *scanf*, *strtod*, *strtol*, *strtoul*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   long l;
   char *lstr = "98765432";

   l = atol(lstr);
   printf("string = %s integer = %ld\n", lstr, l);
   return(0);
}
```

bar

Funktion Zeichnet eine zweidimensionale Leiste.

Syntax

```
#include <graphics.h>
#include <conio.h>
void far bar(int left, int top, int right, int bottom);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *bar* zeichnet eine gefüllte rechteckige Leiste. Verwendet wird die aktuelle Zeichenfarbe sowie das aktuelle Füllmuster. Einen Umriß zeichnet *bar* nicht – hierzu können Sie *bar3d* mit *depth* =0 verwenden.

Die obere linke Ecke der Leiste ist durch die Koordinaten (*left*, *top*) und die untere rechte Ecke durch (*right*, *bottom*) – in Pixel – festgelegt.

Rückgabewert Keiner.

Siehe auch *bar3d*, *rectangle*, *setcolor*, *setfillstyle*, *setlinestyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;

   /* initialize graphics and local
      variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error
       occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error
                  code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i=SOLID_FILL; i<USER_FILL; i++)
   {
      /* set the fill style */
      setfillstyle(i, getmaxcolor());

      /* draw the bar */
      bar(midx-50, midy-50, midx+50,
         midy+50);

      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

bar3d

Funktion — Zeichnet einen dreidimensionalen Balken.

Syntax

```
#include <graphics.h>
void far bar3d(int left, int top, int right,
               int bottom, int depth, int topflag);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Balken, die mit *bar3d* gezeichnet werden, sind »dreidimensional« und bestehen aus einem gefüllten Rechteck, dessen obere linke Ecke durch die Koordinaten (*left, top*) und dessen untere rechte Ecke durch (*right, bottom*) – in Pixel – festgelegt ist. *bar3d* zeichnet zuerst die Umrißlinien in der aktuellen Farbe und der aktuellen Linienart, danach füllt die Funktion die umschlossene Fläche (wie *bar*) mit dem aktuellen Füllmuster aus.

Der Parameter *topflag* legt fest, ob *bar3d* einen oberen Abschluß des Balkens zeichnet (*topflag*=1) oder nicht (*topflag*=0) – im letzteren Fall lassen sich mehrere Balken übereinanderstapeln. Über den Parameter *depth* läßt sich die räumliche Tiefe in Pixel bestimmen. Als Faustregel läßt sich sagen, daß die Tiefe rund 25% der Breite betragen sollte.

Dies ist ein typischer Aufruf von *bar3d*:

```
bar3d(left, top, right, bottom, (right-left) / 4, 1);
```

Rückgabewert — Keiner.

Siehe auch — *bar, rectangle, setcolor, setfillstyle, setlinestyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)

{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;

   /* initialize graphics, local variables*/
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
```

```
   if (errorcode != grOk)  /* an error
       occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i=EMPTY_FILL; i<USER_FILL; i++)
   {
      /* set the fill style */
      setfillstyle(i, getmaxcolor());

      /* draw the 3-d bar */
      bar3d(midx-50, midy-50, midx+50,
            midy+50, 10, 1);

      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

bcd

Funktion

Konvertiert eine Zahl in eine BCD-Zahl (eine binär codierte Dezimalzahl).

Syntax

```
#include <bcd.h>
bcd bcd(int x);
bcd bcd(double x);
bcd bcd(double x, int decimals);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung

Alle gängigen arithmetischen Operatoren sind in *bcd.h* überladen, damit sie mit BCD-Zahlen arbeiten können. BCD-Zahlen haben eine Genauigkeit von 17 Stellen und liegen in einem Bereich zwischen $1*10^{-125}$ und $1*10^{125}$.

Mit der Funktion *real* kann eine BCD-Zahl in einen **float-**, **double-**oder **long double -**Wert zurückkonvertiert werden.

Der Parameter *decimals* ist optional. Mit ihm kann angegeben werden, wieviele dezimale Nachkommastellen in die Konvertierung einbezogen werden sollen.

Die Zahl wird auf die nächste ganze Zahl gerundet, Bruchteile werden auf eine gerade Ziffer gerundet.

Rückgabewert Es wird das BCD-Äquivalent der gegebenen Zahl zurückgeliefert.

Siehe auch *real*

Beispiel

```
#include <iostream.h>
#include <bcd.h>

double x = 10000.0;          // ten thousand dollars
bcd a = bcd(x/3,2);          // a third, rounded to
                             // nearest penny

int main(void)
{
   cout << "share of fortune = $" << a << "\n";
   return 0;
}
```

bdos

Funktion Ruft eine DOS-Funktion auf.

Syntax

```
#include <dos.h>
int bdos(int dosfun, unsigned dosdx, unsigned dosal);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Über diese Funktion lassen sich die meisten DOS-Systemaufrufe direkt programmieren. Spezifische Details zu den einzelnen Funktionen entnehmen Sie bitte dem DOS-Referenzhandbuch.

Für Systemaufrufe, die ein Integerargument erwarten, kann die Funktion *bdos* eingesetzt werden. Wird ein Zeiger erwartet, benutzen Sie *bdosptr*. In den großen Datenmodellen (*compact*, *large*, *huge*) ist es wichtig, *bdosptr* für solche Systemaufrufe einzusetzen, die einen Zeiger als Argument erwarten.

dosfun gibt die Nummer der Funktion an und ist in Ihrem DOS-Referenzhandbuch definiert, *dosdx* legt den Inhalt des Registers DX und *dosal* den Wert des Registers AL fest.

Rückgabewert Zurückgeliefert wird der Wert, auf den das Register AX von DOS gesetzt wurde.

Siehe auch *bdosptr, geninterrupt, int86, int86x, intdos, intdosx*

Beispiel

```
#include <stdio.h>
#include <dos.h>

/* Get current drive as 'A', 'B', ... */
char current_drive(void)
{
   char curdrive;

   /* Get current disk as 0, 1, ... */
   curdrive = bdos(0x19, 0, 0);
   return('A' + curdrive);
}

int main(void)
{
   printf("The current drive is %c:\n",
          current_drive());
   return 0;
}
```

Programmausgabe:

```
The current drive is C:
```

bdosptr

Funktion Ruft eine DOS-Funktion auf.

Syntax

```
#include <dos.h>
int bdosptr(int dosfun, void *argument,
            unsigned dosal);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Über diese Funktion lassen sich die meisten DOS-Systemaufrufe direkt programmieren. Spezifische Details zu den einzelnen Funktionen entnehmen Sie bitte dem DOS-Referenzhandbuch.

Für Systemaufrufe, die ein Integerargument erwarten, kann die Funktion *bdos* eingesetzt werden. Wird ein Zeiger erwartet, benutzen Sie *bdosptr*. In den großen Datenmodellen (*compact*, *large*, *huge*) ist es wichtig, *bdosptr* für solche Systemaufrufe einzusetzen, die einen Zeiger als Argument erwarten. In den kleinen Datenmodellen wird mit *argument* das Register DX gesetzt, in den großen Datenmodellen werden die Werte von DS:DX angegeben, die der Systemaufruf benutzt.

dosfun gibt die Nummer der Funktion an und ist in Ihrem DOS-Referenzhandbuch definiert.

dosal ist der Wert des Registers AL.

Rückgabewert Zurückgeliefert wird der Wert, auf den das Register AX von DOS gesetzt wurde. Tritt bei der Ausführung ein Fehler auf, dann liefert die Funktion den Wert -1 und die globalen Variablen *errno* und *_doserrno* werden entsprechend gesetzt.

Siehe auch *bdos, geninterrupt, int86, int86x, intdos, intdosx*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <dir.h>
#include <dos.h>
#include <errno.h>
#include <stdlib.h>

#define  BUFLEN  80

int main(void)
{
   char  buffer[BUFLEN];
   int   test;

   printf("Enter full pathname of a directory\n");
   gets(buffer);

   test = bdosptr(0x3B,buffer,0);
      if(test)
      {
         printf("DOS error message: %d\n", errno);
         /* See errno.h for error listings */
         exit (1);
      }

   getcwd(buffer, BUFLEN);
   printf("The current directory is: %s\n", buffer);

   return 0;
}
```

bioscom

Funktion Steuert Ein- und Ausgaben über serielle Schnittstellen.

Syntax

```
#include <bios.h>
int bioscom(int cmd, char abyte, int port);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

Diese Funktion ermöglicht das Setzen von Parametern der seriellen Schnittstellen sowie die Ein- und Ausgabe von Daten über die in *port* angegebene Schnittstelle.

Der Parameter *port* legt fest, mit welcher der beiden Schnittstellen gearbeitet wird (0 =COM1, 1 =COM2, usw.).

Der Parameter *cmd* kann Werte von 0 bis 3 annehmen:

0 setzt die Übertragungsparameter auf den in *abyte* enthaltenen Wert;

1 gibt den Wert von *abyte* über die serielle Schnittstelle aus;

2 liest ein Zeichen über die serielle Schnittstelle;

3 liefert den aktuellen Status der seriellen Schnittstelle.

Der Parameter *abyte* ist eine Kombination von Bitwerten (dabei wird aus jeder Gruppe ein Wert ausgewählt):

Bitwert	Bedeutung	Bitwert	Bedeutung
Übertragungsgeschwindigkeit		*Art der Paritätsprüfung*	
0x00	110 Baud	0x00	keine Paritätsprüfung
0x20	150 Baud	0x08	ungerade Parität
0x40	300 Baud	0x18	gerade Parität
0x60	600 Baud	*Stopbits*	
0x80	1200 Baud	0x00	1 Stopbit
0xA0	2400 Baud	0x04	2 Stopbits
0xC0	4800 Baud		
0xE0	9600 Baud	*Datenbits*	
		0x02	7 Datenbits
		0x03	8 Datenbits

Die einzelnen Werte müssen miteinander verbunden werden. Ein Beispiel: die Parameter »9600 Baud, ungerade Parität, 1 Stopbit, 8 Datenbits« werden mit dem Wert 0xEB von *abyte* gesetzt, der sich aus (0xE0 | 0x08 | 0x00 | 0x03) ergibt.

bioscom verwendet den BIOS-Interrupt 0x14.

Rückgabewert

Das höherwertige Byte des von *bioscom* zurückgelieferten Integers enthält für alle Werte von *cmd* eine Statusinformation, die in einzelnen Bits codiert ist, das niederwertige Byte hängt vom Wert von *cmd* ab. Die Bedeutung der höherwertigen Bits:

Bit 15 Time out
Bit 14 Schieberegister für Übertragung leer
Bit 13 Übertragungspuffer leer
Bit 12 Break-Signal erhalten
Bit 11 Framing-Fehler
Bit 10 Paritätsfehler
Bit 09 Überlauf
Bit 08 Daten bereit

Wenn der in *abyte* enthaltene Wert nicht gesendet werden konnte, wird Bit 15 auf 1 gesetzt, in allen anderen Fällen werden die übrigen höher- und niederwertigen Bits in geeigneter Weise gesetzt. Bei einem Framing-Fehler wird beispielsweise das Bit 11 auf 1 gesetzt.

Wenn *cmd* den Wert 2 hat und ein Zeichen fehlerfrei gelesen werden konnte, dann sind sämtliche Bits des höherwertigen Bytes auf »0« gesetzt und das niederwertige Byte enthält das gelesene Zeichen. Im Fall eines Fehlers wird mindestens eines der höherwertigen Bits auf 1 gesetzt.

Für die Werte 0 und 3 von *cmd* liefert *bioscom* einen Wert zurück, dessen höherwertiges Byte sich aus den oben beschriebenen Bits zusammensetzt. Zusätzlich enthält das niederwertige Byte die folgenden Informationen:

Bit 07 Received line signal detect
Bit 06 Ring indicator
Bit 05 Data set ready
Bit 04 Clear to send
Bit 03 Change in receive line signal detector
Bit 02 Trailing edge ring detector
Bit 01 Change in data set ready
Bit 00 Change in clear to send

Beispiel

```
#include <bios.h>
#include <conio.h>

#define COM1       0
#define DATA_READY 0x100
#define TRUE       1
#define FALSE      0

#define SETTINGS ( 0x80 | 0x02 | 0x00 | 0x00)

int main(void)
{
   int in, out, status, DONE = FALSE;

   bioscom(0, SETTINGS, COM1);
   cprintf("... BIOSCOM [ESC] to exit ...\n");
   while (!DONE)
   {
      status = bioscom(3, 0, COM1);
      if (status & DATA_READY)
         if ((out = bioscom(2, 0, COM1) & 0x7F) != 0)
            putch(out);
         if (kbhit())
         {
            if ((in = getch()) == '\x1B')
               DONE = TRUE;
            bioscom(1, in, COM1);
```

```
        }
    }
    return 0;
}
```

_bios_disk

Funktion Direkter Zugriff auf Disketten /Festplatten über BIOS-Interrupt.

Syntax
```
#include <bios.h>
unsigned _bios_disk(unsigned cmd, struct diskinfo_t
                    *dinfo);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Diese Funktion verwendet den BIOS-Interrupt 0x13 für direkte Zugriffe auf Disketten- und Festplattenlaufwerke. Die Art der auszuführenden Operation wird über den Wert von *cmd* festgelegt. *dinfo* zeigt auf die Struktur *diskinfo_t* mit den weiteren Argumenten, die für die Operation benötigt werden.

Die *diskinfo_t*-Struktur ist in bios.h definiert und hat folgendes Format:

```
struct diskinfo_t {
    unsigned drive, head, track, sector, nsectors;
    void far *buffer;
};
```

Der Parameter *drive* legt fest, welches Laufwerk angesprochen werden soll, und wird wie folgt interpretiert:

- Werte im Bereich von 0x00 bis 0x7F sprechen Diskettenlaufwerke an – 0x00 steht für das erste Laufwerk, 0x01 für das zweite usw.
- Werte im Bereich von 0x80 bis 0xFF adressieren Festplattenlaufwerke – 0x80 steht für das erste Festplattenlaufwerk, 0x81 für das zweite usw.

Bei Festplatten wird nicht die Partition sondern das physikalische Laufwerk angegeben. Falls die Festplatte partitioniert ist, muß die Interpretation der Partitionstabelle durch das aufrufende Programm erfolgen.

Ob die übrigen Parameter in der Struktur *diskinfo_t* benötigt werden, hängt vom Wert des *cmd*-Parameters ab.

Für *cmd* sind folgende Werte in bios.h definiert:

_DISK_RESET – führt ein »hartes« Reset des Controllers durch und ignoriert alle anderen Parameter von *diskinfo_t*.

_DISK_STATUS – liefert den Status der zuletzt ausgeführten Operation zurück und ignoriert alle anderen Parameter von *diskinfo_t*.

_DISK_READ – liest einen oder mehrere Sektoren von der Diskette/Festplatte in den Speicher. Der erste zu lesende Sektor ist durch die Parameter *head*, *track* und *sector* festgelegt, die Anzahl der zu lesenden Sektoren durch *nsectors*. Die Daten werden, 512 Bytes pro Sektor, in *buffer* eingelesen. Ist die Operation erfolgreich, ist das höherwertige Byte des zurückgelieferten Werts 0 und das niederwertige Byte enthält die Anzahl der Sektoren. Bei einem Fehler hat das höherwertige Byte des Rückgabewerts einen der folgenden Werte:

0x01	Unbekannter Befehl
0x02	Adreßmarkierung nicht gefunden
0x03	Schreibgeschützt (nur für Diskettenlaufwerke)
0x04	Sektor nicht gefunden/ungültige Sektornummer
0x05	RESET konnte nicht ausgeführt werden (Festplatte)
0x06	Diskettenwechsel seit letzter Operation
0x07	Parametertabelle ungültig
0x08	DMA-Fehler
0x09	Überschreitung der Segmentgrenze von 64 KByte bei DMA-Operation
0x0A	Sektor ist als »bad« markiert
0x0B	Spur ist als »bad« markiert
0x0C	Spur ungültig
0x10	Sektordaten ungültig (CRC/ECC-Prüfsummenfehler)
0x11	Datenfehler, durch CRC/ECC korrigiert (nur Festplatten)
0x20	Controller-Fehler
0x40	Suchen-Operation fehlgeschlagen
0x80	Laufwerk nicht bereit (time out)
0xAA	Platte nicht bereit (Festplattensysteme)
0xBB	Undefinierter Fehler (Festplattensysteme)
0xCC	Schreibfehler
0xEO	Statusfehler
0xFF	Abtastung fehlgeschlagen

Der Wert 0x11 zeigt an sich keinen Fehler an und ist diagnostischer Natur. Die gelesenen Daten sind korrekt bzw. wurden durch den Prüfsummenmechanismus korrigiert.

_DISK_WRITE – schreibt einen oder mehrere Sektoren aus dem Speicher auf die Diskette/Festplatte. Der erste zu schreibende Sektor ist durch die Parameter *head*, *track* und *sector* festgelegt, die Anzahl der zu schreibenden Sektoren durch *nsectorss*. Die Daten werden mit 512 Bytes pro Sektor aus *buffer* geschrieben. Die Rückgabewerte finden Sie in der obigen Beschreibung von *_DISK_READ*.

_DISK_VERIFY – prüft einen oder mehrere Sektoren. Der erste zu überprüfende Sektor ist durch die Parameter *head*, *track* und *sector* festgelegt, die Anzahl der zu überprüfenden Sektoren durch *nsectors*. Die Rückgabewerte finden Sie in der obigen Beschreibung von *_DISK_READ*.

_DISK_FORMAT – formatiert die durch *track* und *head* angegebene Spur. *buffer* zeigt muß in diesem Fall auf eine Tabelle mit Parametern für Sektorzahl, -nummern, -länge usw. zeigen. Lesen Sie die Beschreibung dieser Tabelle und der Formatierung im *Technischen Referenzhandbuch* für den IBM PC nach.

Rückgabewert *_bios_disk* liefert den durch den BIOS-Interrupt 0x13 gesetzten Wert im AX-Register zurück.

Siehe auch *absread*, *abswrite*, *biosdisk*

Beispiel

```
#include <bios.h>
#include <stdio.h>

int main(void)
{
  struct diskinfo_t dinfo;
  int result;
  static char dbuf[512];

  dinfo.drive =  0;    /* drive number for A: */
  dinfo.head  =  0;    /* disk head number */
  dinfo.track =  0;    /* track number */
  dinfo.sector  =  1;  /* sector number */
  dinfo.nsectors =  1; /* sector count */
  dinfo.buffer = dbuf; /* data buffer */

  printf("Attempting to read from drive A:\n");
  result = _bios_disk(_DISK_READ, &dinfo);
  if ((result & 0xff00) == 0)
  {
    printf("Disk read from A: successful.\n");
    printf("First three bytes read are "
           "0x%02x 0x%02x 0x%02x\n",
           dbuf[0] & 0xff, dbuf[1] & 0xff,
           dbuf[2] & 0xff);
  }
  else
    printf("Cannot read drive A, status = 0x%02x\n",
           result);
  return 0;
}
```

biosdisk

Funktion Direkter Zugriff auf Disketten /Festplatten über BIOS-Interrupt.

Syntax
```
#include <bios.h>
int biosdisk(int cmd, int drive, int head, int track,
             int sector, int nsects, void *buffer);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Diese Funktion verwendet den BIOS-Interrupt 0x13 für direkte Zugriffe auf Disketten- und Festplattenlaufwerke.

Der Parameter *drive* legt fest, welches Laufwerk angesprochen werden soll, und wird wie folgt interpretiert:

- Werte im Bereich von 0x00 bis 0x7F sprechen Diskettenlaufwerke an – 0x00 steht für das erste Laufwerk, 0x01 für das zweite usw.
- Werte im Bereich von 0x80 bis 0xFF adressieren Festplattenlaufwerke – 0x80 steht für das erste Festplattenlaufwerk, 0x81 für das zweite usw.

Bei Festplatten wird nicht die Partition sondern das physikalische Laufwerk angegeben. Falls die Festplatte partitioniert ist, muß die Interpretation der Partitionstabelle durch das aufrufende Programm erfolgen.

Der Parameter *cmd* legt die durchzuführende Operation fest. Ob die übrigen Parameter benötigt werden, hängt vom Wert des *cmd*-Parameters ab.

Für *cmd* sind auf allen Computern der IBM PC-Serie und Kompatiblen die folgenden Werte möglich:

0 führt ein »hartes« Reset des Controllers durch und ignoriert sämtliche anderen Parameter.

1 liefert den Status der zuletzt ausgeführten Operation zurück und ignoriert alle anderen Parameter.

2 liest einen oder mehrere Sektoren von der Diskette/Festplatte in den Speicher. Der erste zu lesende Sektor ist durch die Parameter *head*, *track* und *sector* festgelegt, die Anzahl der zu lesenden Sektoren durch *nsects*. Die Daten werden mit 512 Bytes pro Sektor in *buffer* eingelesen.

3 schreibt einen oder mehrere Sektoren aus dem Speicher auf die Diskette/Festplatte. Der erste zu schreibende Sektor ist durch die Parameter *head*, *track* und *sector* festgelegt, die Anzahl der zu schreibenden Sektoren durch *nsects*. Die Daten werden mit 512 Bytes pro Sektor aus *buffer* geschrieben.

4 prüft einen oder mehrere Sektoren. Der erste zu überprüfende Sektor ist durch die Parameter *head*, *track* und *sector* festgelegt, die Anzahl der zu überprüfenden Sektoren durch *nsects*.

5 formatiert die durch *track* und *head* angegebene Spur. *buffer* muß in diesem Fall auf eine Tabelle mit Parametern für Sektorzahl, -nummern, -länge usw. zeigen. Lesen Sie die Beschreibung dieser Tabelle und der Formatierung im *Technischen Referenzhandbuch* für den IBM PC nach.

Die folgenden Werte für *cmd* sind nur auf einem XT, AT, PS/2 oder Kompatiblen verwendbar:

6 formatiert die durch *track* und *head* angegebene Spur und trägt Kennzeichenbits für defekte Sektoren ein;

7 formatiert ab einer bestimmten Spur;

8 liefert die Parameter des aktuellen Laufwerks in den ersten vier Bytes von *buffer* zurück;

9 initialisiert die Parametertabellen für Festplatten;

10 long read – liest 512 Bytes plus 4 zusätzliche Prüfbytes pro Sektor;

11 long write – schreibt 512 Bytes plus 4 zusätzliche Prüfbytes pro Sektor;

12 seek – setzt den Schreib/Lesekopf auf eine bestimmte Spur;

13 modifiziert den Reset;

14 liest den Sektorpuffer;

15 schreibt den Sektorpuffer;

16 prüft, ob das angegebene Laufwerk betriebsbereit ist;

17 kalibriert das Laufwerk neu;

18 Test des Controller-RAM;

19 Test des Laufwerks;

20 Test der internen Controller-Funktionen.

Achtung: biosdisk arbeitet unterhalb der Dateiebene direkt mit den Sektoren. *Die Funktion kann Dateiinhalte und Verzeichnisse auf einer Festplatte zerstören.*

Rückgabewert Sämtliche Aufrufe von *biosdisk* liefern ein Statusbyte zurück, für das die folgenden Werte definiert sind:

0x00	Fehlerfreie Ausführung
0x01	Unbekannter Befehl (ungültiger Wert von *cmd*)
0x02	Adreß-Kennzeichnung nicht gefunden
0x03	Schreibgeschützt (nur für Diskettenlaufwerke)

0x04	Sektor nicht gefunden/ungültige Sektornummer
0x05	RESET konnte nicht ausgeführt werden (Festplatte)
0x06	Diskettenwechsel seit letzter Operation
0x07	Parametertabelle ungültig
0x08	DMA-Fehler
0x09	Überschreitung der Segmentgrenze von 64 KByte bei DMA-Operation
0x0A	Sektor ist als »bad« markiert
0x0B	Spur ist als »bad« markiert
0x0C	Spur ungültig
0x10	Sektordaten ungültig (CRC/ECC-Prüfsummenfehler)
0x11	Datenfehler, durch CRC/ECC korrigiert (nur Festplatten)
0x20	Controller-Fehler
0x40	Suchen-Operation fehlgeschlagen
0x80	Laufwerk nicht bereit (time out)
0xAA	Platte nicht bereit (Festplattensysteme)
0xBB	Undefinierter Fehler (Festplattensysteme)
0xCC	Schreibfehler
0xEO	Statusfehler
0xFF	Abtastung fehlgeschlagen

Der Wert 0x11 zeigt an sich keinen Fehler an und ist diagnostischer Natur.

Die gelesenen Daten sind korrekt bzw. wurden durch den Prüfsummenmechanismus korrigiert.

Siehe auch *absread, abswrite*

Beispiel

```
#include <bios.h>
#include <stdio.h>

int main(void)
{
   int result;
   char buffer[512];

   printf("Testing to see if drive a: is ready\n");
   result = biosdisk(4,0,0,0,0,1,buffer);
   result &= 0x02;
   (result) ? (printf("Drive A: Ready\n")) :
              (printf("Drive A: Not Ready\n"));

   return 0;
}
```

biosequip

Funktion Liefert eine Liste installierter Peripheriegeräte.

Syntax

```
#include <bios.h>
int biosequip(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Diese Funktion benutzt den BIOS-Interrupt 0x11 zur Ermittlung der Systemkonfiguration.

Rückgabewert Der zurückgelieferte Wert setzt sich aus mehreren Bitfeldern zusammen. Die Werte für IBM PC und Kompatible:

Bit 14-15: Anzahl der parallel angeschlossenen Drucker
00 0 Drucker
01 1 Drucker
10 2 Drucker
11 3 Drucker

Bit 13: seriell angeschlossener Drucker
Bit 12: Gameport angeschlossen

Bit 9-11: Anzahl der seriellen Schnittstellen
000 0 Schnittstellen
001 1 Schnittstelle
010 2 Schnittstellen
011 3 Schnittstellen
100 4 Schnittstellen
101 5 Schnittstellen
110 6 Schnittstellen
111 7Schnittstellen

Hinweis: DOS erkennt normalerweise nur 2 serielle Schnittstellen, kann aber zur Erkennung von 4 Schnittstellen gezwungen werden, der IBM PS/2 kann bis zu 8 Schnittstellen erkennen.

Bit 8: DMA »direct memory access«
0 Rechner hat DMA
1 Rechner hat kein DMA, z.B. PC Jr.

Bit 6-7: Anzahl von Laufwerken
00 1 Laufwerk
01 2 Laufwerke
10 3 Laufwerke
11 4 Laufwerke, nur falls Bit 0 den Wert 1 hat

Bit 4-5: Video-Modus beim Start
00 unbenutzt
01 40x25 schwarzweiß mit Farbkarte
10 80x25 schwarzweiß mit Farbkarte
11 80x25 schwarzweiß mit Monochromkarte

Bit 2-3: RAM-Größe auf Hauptplatine

00 16 K
01 32 K
10 48 K
11 64 K

Bit 1: Mathematischer Coprozessor installiert
Bit 0: Von Diskette booten

Beispiel

```
#include <bios.h>
#include <stdio.h>

#define CO_PROCESSOR_MASK 0x0002

int main(void)
{
   int equip_check;

   /* get the current equipment configuration */
   equip_check = biosequip();

   /* check if there is a coprocessor installed */
   if (equip_check & CO_PROCESSOR_MASK)
      printf("There is a math coprocessor "
             "installed.\n");
   else
      printf("No math coprocessor installed.\n");
   return 0;
}
```

_bios_equiplist

Funktion Liefert eine Liste installierter Peripheriegeräte.

Syntax

```
#include <bios.h>
unsigned _bios_equiplist(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_bios_equiplist* benutzt den BIOS-Interrupt 0x11 zur Ermittlung der Systemkonfiguration.

Rückgabewert Der zurückgelieferte Wert setzt sich aus mehreren Bitfeldern zusammen. Die Werte für IBM PC und Kompatible:

Bit 14-15: Anzahl der parallel angeschlossenen Drucker
00 0 Drucker
01 1 Drucker
10 2 Drucker
11 3 Drucker

Bit 13: seriell angeschlossener Drucker

Bit 12: Gameport angeschlossen

Bit 9-11: Anzahl der seriellen Schnittstellen
000 0 Schnittstellen
001 1 Schnittstelle
010 2 Schnittstellen
011 3 Schnittstellen
100 4 Schnittstellen
101 5 Schnittstellen
110 6 Schnittstellen
111 7 Schnittstellen

Hinweis: DOS erkennt normalerweise nur 2 serielle Schnittstellen, kann aber zur Erkennung von 4 Schnittstellen gezwungen werden, der IBM PS/2 kann bis zu 8 Schnittstellen erkennen.

Bit 8: DMA »direct memory access«
0 Rechner hat DMA
1 Rechner hat kein DMA, z.B. PC Jr.

Bit 6-7: Anzahl der Laufwerke
00 1 Laufwerk
01 2 Laufwerke
10 3 Laufwerke
11 4 Laufwerke, nur falls Bit 0 den Wert 1 hat

Bit 4-5: Video-Modus beim Start
00 unbenutzt
01 40x25 schwarzweiß mit Farbkarte
10 80x25 schwarzweiß mit Farbkarte
11 80x25 schwarzweiß mit Monochromkarte

Bit 2-3: RAM-Größe auf Hauptplatine
00 16 K
01 32 K
10 48 K
11 64 K

Bit 1: Mathematischer Coprozessor installiert
Bit 0: Von Diskette booten

Beispiel

```
#include <stdio.h>
#include <bios.h>

#define CO_PROCESSOR_MASK 0x0002
int main(void)
{
  unsigned equip_check;
  /* get the current equipment configuration */
  equip_check = _bios_equiplist();
```

```
  /* check if there is a coprocessor installed */
  if (equip_check & CO_PROCESSOR_MASK)
    printf("There is a math coprocessor "
           "installed.\n");
   else
    printf("No math coprocessor installed.\n");
  return 0;
}
```

bioskey

Funktion Tastaturabfrage über BIOS-Interrupt.

Syntax

```
#include <bios.h>
int bioskey(int cmd);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Diese Funktion verwendet den BIOS-Interrupt 0x16 zur Kommunikation mit der Tastatur. Die Art der auszuführenden Operation wird über den Wert von *cmd* festgelegt.

Rückgabewert Der Rückgabewert hängt von der Operation ab, die über den Wert von *cmd* festgelegt wurde.

0 Falls die niederwertigen 8 Bit ungleich Null sind, liefert *bioskey* das nächste ASCII-Zeichen aus dem Tastaturpuffer oder wartet auf die Eingabe eines Zeichens und liefert dieses Zeichen zurück. Bei der Eingabe eines »normalen« Zeichens liefert *bioskey* den entsprechenden ASCII-Code im niederwertigen Byte. Für Spezialzeichen wie Funktions- und Cursortasten hat das niederwertige Byte den Wert 0, das höherwertige Byte enthält den erweiterten Tastaturcode (siehe *Technisches Referenzhandbuch* für den IBM PC).

1 prüft, ob ein Zeichen anliegt. Wenn nicht, wird der Wert 0 zurückgeliefert, ansonsten der entsprechende Tastencode. Ein Rückgabewert von 0xFFFF(-1) bedeutet, daß Strg Untbr gedrückt wurde. Über diese Funktion gelesene Zeichen werden nicht »abgeholt« – sie verbleiben im Tastaturpuffer und werden nach dem nächsten Aufruf von *bioskey* mit dem *cmd*-Wert 0 ausgegeben.

2 liefert den aktuellen Zustand der Umschalttasten (Shift, Alt usw.). Der Rückgabewert ergibt sich aus den über ODER-Operationen verbundenen Bitwerten:

Bit 7	0x80	Einfg aktiv
Bit 6	0x40	0 aktiv

Bit 5	0x20	NumLock aktiv
Bit 4	0x10	Rollen aktiv
Bit 3	0x08	Alt gedrückt
Bit 2	0x04	Strg gedrückt
Bit 1	0x02	⇧ links gedrückt
Bit 0	0x01	⇧ rechts gedrückt

Beispiel

```
#include <stdio.h>
#include <bios.h>
#include <ctype.h>

#define RIGHT  0x01
#define LEFT   0x02
#define CTRL   0x04
#define ALT    0x08

int main(void)
{
   int key, modifiers;

   /* function 1 returns 0 until a key is pressed */
   while (bioskey(1) == 0);

   /* function 0 returns the key that is waiting */
   key = bioskey(0);

   /* use function 2 to determine */
   /* if shift keys were used */
   modifiers = bioskey(2);
   if (modifiers)
   {
      printf("[");
      if (modifiers & RIGHT) printf("RIGHT");
      if (modifiers & LEFT)  printf("LEFT");
      if (modifiers & CTRL)  printf("CTRL");
      if (modifiers & ALT)   printf("ALT");
      printf("]");
   }
   /* print out the character read */
   if (isalnum(key & 0xFF))
      printf("'%c'\n", key);
   else
      printf("%#02x\n", key);
   return 0;
}
```

_bios_keybrd

Funktion Tastaturabfrage über BIOS-Interrupt.

Syntax
```
#include <bios.h>
unsigned _bios_keybrd(unsigned cmd);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Die Funktion *_bios_keybrd* verwendet den BIOS-Interrupt 0x16 zur Kommunikation mit der Tastatur. Die Art der auszuführenden Operation wird über den Wert von *cmd* festgelegt.

Rückgabewert Der Rückgabewert von *_bios_keybrd* hängt von der Operation ab, die über den Wert von *cmd* festgelegt wurde.

Folgende Werte sind für *cmd* in bios.h definiert:

_KEYBRD_READ – falls die niederwertigen 8 Bit ungleich 0 sind, liefert *_bios_keybrd* das nächste Zeichen aus dem Tastaturpuffer oder wartet auf die Eingabe eines Zeichens und liefert dieses Zeichen zurück. Bei der Eingabe eines »normalen« Zeichens liefert *_bios_keybrd* den entsprechenden ASCII-Code im niederwertigen Byte. Für Spezialzeichen wie Funktions- und Pfeiltasten hat das niederwertige Byte den Wert 0, das höherwertige Byte enthält den erweiterten Tastaturcode (siehe *Technisches Referenzhandbuch* für den IBM PC).

_NKEYBRD_READ – verwenden Sie diesen Wert anstelle von *_KEYBRD_READ*, wenn Zeichen einer erweiterten Tastatur abgerufen werden sollen. Eine erweiterte Tastatur verfügt über zusätzliche Pfeil- und Funktionstasten.

_KEYBRD_READY – prüft, ob ein Zeichen anliegt. Wenn dies nicht der Fall ist, wird der Wert 0 zurückgeliefert, ansonsten derselbe Wert wie für _KEYBRD_READ. Ein Rückgabewert von 0xFFFF(-1) bedeutet, daß Strg Untbr gedrückt wurde. Über diese Funktion gelesene Zeichen werden nicht »abgeholt« – sie verbleiben im Tastaturpuffer und werden durch den nächsten Aufruf von *_bios_keybrd* mit dem *cmd*-Wert _KEYBRD_READ oder _NKEYBRD_READ ausgegeben.

_NKEYBRD_READY – verwenden Sie diesen Wert für die Überprüfung des Status einer erweiterten Tastatur. Eine erweiterte Tastatur verfügt über zusätzliche Pfeil- und Funktionstasten.

_KEYBRD_SHIFTSTATUS – liefert den aktuellen Zustand der Umschalttasten (Shift , Alt usw.).

Der Rückgabewert ergibt sich aus den über ODER-Operationen verbundenen Bitwerten:

Bit 7 0x80 Einfg aktiv

Bit 6	0x40	⇩ aktiv
Bit 5	0x20	Num aktiv
Bit 4	0x10	Rollen aktiv
Bit 3	0x08	Alt gedrückt
Bit 2	0x04	Strg gedrückt
Bit 1	0x02	⇧ links gedrückt
Bit 0	0x01	⇧ rechts gedrückt

_NKEYBRD_SHIFTSTATUS – verwenden Sie diesen Wert anstelle von _KEYBRD_SHIFTSTATUS, um den aktuellen Status der Umschalttasten (Shift, Alt usw.) von erweiterten Tastaturen festzustellen.

Der Rückgabewert ergibt sich aus den über ODER-Operationen verbundenen Bitwerten von _KEYBRD_SHIFTSTATUS und den folgenden Werten:

Bit 15	0x8000	s-Abf gedrückt
Bit 14	0x4000	⇩ gedrückt
Bit 13	0x2000	NumLock gedrückt
Bit 12	0x1000	Rollen gedrückt
Bit 11	0x0800	Alt rechts gedrückt
Bit 10	0x0400	Strg rechts gedrückt
Bit 9	0x0200	Alt links gedrückt
Bit 8	0x0100	Strg links gedrückt

Beispiel

```
#include <stdio.h>
#include <bios.h>
#include <ctype.h>

#define RIGHT  0x01
#define LEFT   0x02
#define CTRL   0x04
#define ALT    0x08

int main(void)
{
  int key, modifiers;

  /* Wait until a key is pressed */
  while (_bios_keybrd(_KEYBRD_READY) == 0);

  /* Fetch the key that is waiting */
  key = _bios_keybrd(_KEYBRD_READ);

  /* Determine if shift keys are used */
  modifiers = _bios_keybrd(_KEYBRD_SHIFTSTATUS);
  if (modifiers)
  {
    printf("[");
```

```
    if (modifiers & RIGHT) printf("RIGHT");
    if (modifiers & LEFT)  printf("LEFT");
    if (modifiers & CTRL)  printf("CTRL");
    if (modifiers & ALT)   printf("ALT");
    printf("]");
  }

  /* print out the character read */
  if (isalnum(key & 0xFF))
    printf("'%c'\n", key);
  else
    printf("%#02x\n", key);
  return 0;
}
```

biosmemory

Funktion — Liefert die Größe des Hauptspeichers zurück.

Syntax

```
#include <bios.h>
int biosmemory(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — Diese Funktion verwendet den BIOS-Interrupt 0x12 zur Bestimmung der RAM-Größe. Zusätzliches RAM in Grafikkarten, Erweiterungsspeicher und Expansionsspeicher wird nicht berücksichtigt.

Rückgabewert — *biosmemory* liefert die Größe des Hauptspeichers in 1-KByte-Blöcken zurück.

Beispiel

```
#include <stdio.h>
#include <bios.h>

int main(void)
{
   int memory_size;

   memory_size = biosmemory();  /* returns value */
                                /* up to 640K */
   printf("RAM size = %dK\n",memory_size);
   return 0;
}
```

_bios_memsize

Funktion Liefert die Größe des Hauptspeichers zurück.

Syntax
```
#include <bios.h>
unsigned _bios_memsize(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Diese Funktion verwendet den BIOS-Interrupt 0x12 zur Bestimmung der RAM-Größe. Zusätzliches RAM in Grafikkarten, Erweiterungsspeicher und Expansionsspeicher wird nicht berücksichtigt.

Rückgabewert *_bios_memsize* liefert die Größe des Hauptspeichers in 1-KByte-Blöcken zurück.

Beispiel
```
#include <stdio.h>
#include <bios.h>

int main(void)
{
  unsigned   memory_size;

  memory_size = _bios_memsize();   /* returns value */
                                   /* up to 640K */

  printf("RAM size = %dK\n", memory_size);

  return 0;
}
```

biosprint

Funktion Kommunikation mit einem Drucker über BIOS-Interrupt.

Syntax
```
#include <bios.h>
int biosprint(int cmd, int abyte, int port);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Diese Funktion verwendet den BIOS-Interrupt 0x17 zur Kommunikation mit einem angeschlossenen Drucker.

Der Wert von *port* legt fest, welche der parallelen Schnittstellen angesprochen wird: 0 steht für LPT1, 1 für LPT2, usw.

Der Wert von *cmd* legt die Art der Aktion fest:

0 Ausgabe des Zeichens in *abyte*;
1 Initialisierung der Schnittstelle;
2 Abfrage des Drucker-Status.

Der Wert von *abyte* kann zwischen 0 und 255 liegen.

Rückgabewert Alle drei Operationen liefern den aktuellen Status des angesprochenen Druckers. Der Rückgabewert ergibt sich aus den über ODER-Operationen verbundenen Bitwerten. Die Bedeutung der einzelnen Bits im gesetzten Zustand:

Bit 0	0x01	Timeout
Bit 3	0x08	I/O-Fehler
Bit 4	0x10	Drucker ist ausgewählt
Bit 5	0x20	Papierende
Bit 6	0x40	Bestätigung
Bit 7	0x80	Drucker wartet

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

int main(void)
{
   #define STATUS  2    /* printer status command */
   #define PORTNUM 0    /* port number for LPT1 */

   int status, abyte=0;

   printf("Please turn off your printer. "
          "Press any key to continue\n");
   getch();
   status = biosprint(STATUS, abyte, PORTNUM);
   if (status & 0x01)
      printf("Device time out.\n");
   if (status & 0x08)
      printf("I/O error.\n");

   if (status & 0x10)
      printf("Selected.\n");
   if (status & 0x20)
      printf("Out of paper.\n");

   if (status & 0x40)
      printf("Acknowledge.\n");
   if (status & 0x80)
      printf("Not busy.\n");

   return 0;
}
```

_bios_printer

Funktion Kommunikation mit einem Drucker über BIOS-Interrupt.

Syntax
```
#include <bios.h>
unsigned _bios_printer(int cmd, int port, int abyte);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Die Funktion *_bios_printer* verwendet den BIOS-Interrupt 0x17 zur Kommunikation mit einem angeschlossenen Drucker.

Der Wert von *port* legt fest, welche der parallelen Schnittstellen angesprochen wird: 0 steht für LPT1, 1 für LPT2, usw.

Der Wert von *cmd* legt die Art der Aktion fest. Folgende Werte sind in bios.h definiert:

_PRINTER_WRITE	Ausgabe des Zeichens in *abyte*; der Wert von *abyte* kann zwischen 0 und 255 liegen;
_PRINTER_INIT	Initialisierung der Schnittstelle; der Parameter *abyte* wird ignoriert;
_PRINTER_STATUS	Abfrage des Drucker-Status; der Parameter *abyte* wird ignoriert.

Rückgabewert Alle drei Operationen liefern den aktuellen Status des angesprochenen Druckers. Der Rückgabewert ergibt sich aus den über ODER-Operationen verbundenen Bitwerten. Die Bedeutung der einzelnen Bits im gesetzten Zustand:

Bit 0	0x01	Timeout
Bit 3	0x08	I/O-Fehler
Bit 4	0x10	Drucker ist selektiert
Bit 5	0x20	Papierende
Bit 6	0x40	Bestätigung
Bit 7	0x80	Drucker wartet

Beispiel
```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

#define PORTNUM 1

int main(void)
{
  unsigned status, abyte=0;
  printf("Please turn off your printer. "
         "Press any key to continue\n");
  getch();
  status = _bios_printer(_PRINTER_STATUS, PORTNUM,
```

```
                                abyte);
  if (status & 0x01)
    printf("Device time out.\n");
  if (status & 0x08)
    printf("I/O error.\n");
  if (status & 0x10)
    printf("Selected.\n");
  if (status & 0x20)
    printf("Out of paper.\n");
  if (status & 0x40)
    printf("Acknowledge.\n");
  if (status & 0x80)
    printf("Not busy.\n");
  return 0;
}
```

_bios_serialcom

Funktion Steuert Ein- und Ausgaben über serielle Schnittstellen.

Syntax
```
#include <bios.h>
unsigned _bios_serialcom(int cmd, int port, char abyte);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Die Funktion *_bios_serialcom* ermöglicht das Setzen von Parametern der RS-232-Schnittstellen sowie die Ein- und Ausgabe von Daten über die in *port* angegebene Schnittstelle.

Der Parameter *port* legt fest, mit welcher der Schnittstellen gearbeitet wird (0 = COM1, 1 = COM2, usw.).

Der Parameter *cmd* kann folgende in bios.h definierten Werte annehmen:

_COM_INIT	setzt die Übertragungsparameter auf den in *abyte* enthaltenen Wert;
_COM_SEND	gibt den Wert von *abyte* über die serielle Schnittstelle aus;
_COM_RECEIVE	liest ein Zeichen über die serielle Schnittstelle; der Wert von *abyte* wird ignoriert;
_COM_STATUS	liest den aktuellen Status der seriellen Schnittstelle; der Wert von *abyte* wird ignoriert.

Wenn der Parameter *cmd* den Wert _COM_INIT hat, wird *abyte* als ODER-Kombination von Bitwerten interpretiert (dabei wird aus jeder Gruppe ein Wert ausgewählt):

Wert	Bedeutung
Datenbits	
_COM_CHR7	7 Datenbits
_COM_CHR7	8 Datenbits
Stopbits	
_COM_STOP1	1 Stopbit
_COM_STOP2	2 Stopbits
Art der Paritätsprüfung	
_COM_NOPARITY	keine Paritätsprüfung
_COM_ODDPARITY	ungerade Parität
_COM_EVENPARITY	gerade Parität
Übertragungsgeschwindigkeit	
_COM_110	110 Baud
_COM_150	150 Baud
_COM_300	300 Baud
_COM_600	600 Baud
_COM_1200	1200 Baud
_COM_2400	2400 Baud
_COM_4800	4800 Baud
_COM_9600	9600 Baud

Die einzelnen Werte müssen durch eine ODER-Operation miteinander verbunden werden. Ein Beispiel: die Parameter »9600 Baud, ungerade Parität, 1 Stopbit, 8 Datenbits« werden mit dem Wert _COM_INIT von *abyte* gesetzt, der sich aus (_COM_9600 | _COM_ODDPARITY | _COM_STOP1 | _COM_CHR8) ergibt.

_bios_serialcom verwendet den BIOS-Interrupt 0x14.

Rückgabewert Das höherwertige Byte des von *_bios_serialcom* zurückgelieferten 16-Bit-Integers enthält für alle Werte von *cmd* eine Statusinformation, die in einzelnen Bits codiert ist, das niederwertige Byte hängt von *cmd* ab. Die Bedeutung der höherwertigen Bits:

Bit 15 Time out
Bit 14 Schieberegister für Übertragung ist leer
Bit 13 Übertragungspuffer leer
Bit 12 Break-Signal erhalten
Bit 11 Framing-Fehler
Bit 10 Paritätsfehler
Bit 09 Überlauf
Bit 08 Daten bereit

Wenn das in *abyte* enthaltene Zeichen nicht gesendet werden konnte, wird Bit 15 auf 1 gesetzt. In allen anderen Fällen werden die übrigen höher- und niederwertigen Bits in geeig-

neter Weise gesetzt. Bei einem Framing-Fehler wird beispielsweise das Bit 11 auf 1 gesetzt.

Wenn *cmd* den Wert _COM_RECEIVE hat und ein Zeichen fehlerfrei gelesen werden konnte, enthalten die niederwertigen Bits das gelesene Zeichen. Im Fall eines Fehlers wird mindestens eines der höherwertigen Bits auf 1 gesetzt. Ist keines der höherwertigen Bits auf 1 gesetzt, wurde das Zeichen fehlerfrei gelesen.

Hat *cmd* den Wert _COM_INIT oder _COM_STATUS, liefert *_bios_serialcom* einen Wert zurück, dessen höherwertiges Byte sich aus den oben beschriebenen Bits zusammensetzt. Zusätzlich enthält das niederwertige Byte die folgenden Informationen:

Bit 07	Received line signal detect
Bit 06	Ring indicator
Bit 05	Data set ready
Bit 04	Clear to send
Bit 03	Change in receive line signal detector
Bit 02	Trailing edge ring detector
Bit 01	Change in data set ready
Bit 00	Change in clear to send

Beispiel

```
#include <bios.h>
#include <conio.h>

#define COM1       0
#define DATA_READY 0x100
#define TRUE       1
#define FALSE      0
#define SETTINGS (_COM_1200 | _COM_CHR7 |
                  _COM_STOP1 | _COM_NOPARITY)

int main(void)
{
  unsigned in, out, status;

  _bios_serialcom(_COM_INIT, COM1, SETTINGS);
  cprintf("... _BIOS_SERIALCOM [ESC] to exit...\r\n");
  for (;;)
  {
    status = _bios_serialcom(_COM_STATUS, COM1, 0);
    if (status & DATA_READY)
      if ((out = _bios_serialcom(_COM_RECEIVE,
          COM1, 0) & 0x7F) != 0)
        putch(out);
    if (kbhit())
    {
      if ((in = getch()) == '\x1B')
```

```
            break;
          _bios_serialcom(_COM_SEND, COM1, in);
        }
      }
      return 0;
    }
```

biostime

Funktion

Liefert den Stand des BIOS-Timer zurück oder setzt einen neuen Wert.

Syntax

```
#include <bios.h>
long biostime(int cmd, long newtime);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

Der BIOS-Timer ist ein eingebauter Zähler, der jeweils um 0 Uhr 00 auf den Wert 0 gesetzt und von da an rund 18.2 mal pro Sekunde um eins erhöht wird. *biostime* verwendet den BIOS-Interrupt 0x1A zum Lesen bzw. Setzen dieses Zählers.

Wenn *cmd* den Wert 0 hat, liefert *biostime* den aktuellen Wert des Zählers zurück. Wenn *cmd* den Wert 1 hat, setzt *biostime* den Zähler auf den durch *newtime* angegebenen **long**-Wert.

Rückgabewert

Wenn *cmd* den Wert 0 hat, liefert *biostime* den aktuellen Wert des Zählers zurück.

Beispiel

```
#include <stdio.h>
#include <bios.h>
#include <time.h>
#include <conio.h>

int main(void)
{
   long bios_time;

   clrscr();
   cprintf("The number of clock ticks "
           "since midnight is:\r\n");
   cprintf("The number of seconds "
           since midnight is:\r\n");
   cprintf("The number of minutes "
           "since midnight is:\r\n");
   cprintf("The number of hours "
           "since midnight is:\r\n");
   cprintf("\r\nPress any key to quit:");
```

```
    while(!kbhit())
    {
       bios_time = biostime(0, 0L);

       gotoxy(50, 1);
       cprintf("%lu", bios_time);

       gotoxy(50, 2);
       cprintf("%.4f", bios_time / CLK_TCK);

       gotoxy(50, 3);
       cprintf("%.4f", bios_time / CLK_TCK / 60);

       gotoxy(50, 4);
       cprintf("%.4f", bios_time / CLK_TCK / 3600);
    }
    return 0;
}
```

_bios_timeofday

Funktion

Liefert den Stand des BIOS-Timers zurück oder setzt einen neuen Wert.

Syntax

```
#include <bios.h>
unsigned _bios_timeofday(int cmd, long *timep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

Der BIOS-Timer ist ein eingebauter Zähler, der jeweils um 0 Uhr 00 auf den Wert 0 gesetzt und von da an rund 18.2 mal pro Sekunde um eins erhöht wird. *_bios_timeofday* verwendet den BIOS-Interrupt 0x1A zum Lesen bzw. Setzen dieses Zählers.

Der Parameter *cmd* kann einen der folgenden Werte haben:

_TIME_GETCLOCK *_bios_timeofday* speichert den aktuellen Stand des Zählers an der Position, auf die *timep* zeigt. Wurde der Zähler seit 0 Uhr 00 nicht gesetzt oder gelesen, liefert *_bios_timeofday* 1 zurück, andernfalls ist das Ergebnis 0.

_TIME_SETCLOCK *_bios_timeofday* setzt den Stand des Zählers auf den durch *timep* bezeichneten **long**-Wert. Die Funktion liefert keinen Wert zurück.

Rückgabewert

Die Funktion *_bios_timeofday* liefert den Wert zurück, auf den das Register AX durch die Abfrage des BIOS-Timer gesetzt wurde.

Beispiel

```
#include <bios.h>
#include <time.h>
#include <conio.h>

int main(void)
{
  long bios_time;
  clrscr();
  cprintf("The number of clock ticks "
          "since midnight is:\r\n");
  cprintf("The number of seconds "
          "since midnight is:\r\n");
  cprintf("The number of minutes "
          "since midnight is:\r\n");
  cprintf("The number of hours "
          "since midnight is:\r\n");
  cprintf("\r\nPress any key to quit:");
  while(!kbhit())
  {
    _bios_timeofday(_TIME_GETCLOCK, &bios_time);
    gotoxy(50, 1);
    cprintf("%lu", bios_time);
    gotoxy(50, 2);
    cprintf("%.4f", bios_time / CLK_TCK);
    gotoxy(50, 3);
    cprintf("%.4f", bios_time / CLK_TCK / 60);
    gotoxy(50, 4);
    cprintf("%.4f", bios_time / CLK_TCK / 3600);
  }
  return 0;
}
```

brk

Funktion Ändert die Speicherbelegung für das Datensegment.

Syntax

```
#include <alloc.h>
int brk(void *addr);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x			

Beschreibung *brk* ändert die Speichergröße, die vom Heap des aufrufenden Programms belegt wird. Die Änderung wird durchgeführt, indem der *Break*-Wert des Programms neu gesetzt wird. Der *Break*-Wert ist die Adresse der ersten Speicherstelle nach dem Ende des Datensegments. Durch Inkrementieren des *Break*-Wertes wird die Größe des belegten Speichers erhöht.

brk setzt den *Break*-Wert auf *addr* und ändert die Speicherbelegung entsprechend. Falls nicht genügend Speicher zur Verfügung steht, ändert die Funktion nichts an der Speicherbelegung.

Rückgabewert Nach erfolgreicher Änderung wird 0 zurückgeliefert. Im Fall eines Fehlers ist der Rückgabewert -1, das Datensegment bleibt unverändert und die globale Variable *errno* bekommt den Wert

ENOMEM Nicht genug Hauptspeicher

Siehe auch *coreleft, sbrk*

Beispiel

```
#include <stdio.h>
#include <alloc.h>

int main(void)
{
   char *ptr;

   printf("Changing allocation with brk()\n");
   ptr = (char *) malloc(1);
   printf("Before brk() call: %lu bytes free\n",
          coreleft());
   brk(ptr+1000);
   printf(" After brk() call: %lu bytes free\n",
          coreleft());
   return 0;
}
```

bsearch

Funktion Binäres Absuchen eines Array.

Syntax

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base,
              size_t nelem, size_t width,
              int (*fcmp)(const void *, const void *));
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *bsearch* sucht eine Tabelle (ein Array) mit *nelem* Elementen ab. Die Elemente müssen sortiert sein. *bsearch* liefert die Adresse des ersten Eintrags in der Tabelle, der mit dem Suchschlüssel übereinstimmt oder 0, wenn keine Übereinstimmung gefunden wird. Da es sich um eine binäre Suche handelt, ist die erste Fundstelle nicht notwendigerweise der erste Eintrag in der Tabelle.

Der Parameter *base* muß auf das erste Element des Array

zeigen, *nelem* ist ein Integer, der die Anzahl der Array-Elemente enthält. *width* gibt die Größe eines einzelnen Elements in Bytes an. Der Datentyp *size_t* ist in stdlib.h als **unsigned int** definiert.

Der Parameter **fcmp* zeigt auf eine vom Programmierer erstellte Routine, der jeweils zwei Argumente in Form von Zeigern (*elem1*, *elem2*) zum Vergleich übergeben werden: Die Routine vergleicht **elem1* und **elem2* und liefert einen Integerwert mit folgenden möglichen Werten zurück:

<0	falls $*elem1 < *elem2$
$==0$	falls $*elem1 == *elem2$
>0	falls $*elem1 > *elem2$

Rückgabewert *bsearch* liefert die Adresse des ersten Elements des Array zurück, das mit den gesuchten Daten übereinstimmt.

Falls der gesuchte Eintrag nicht existiert, wird 0 zurückgeliefert.

Siehe auch *lfind*, *lsearch*, *qsort*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

typedef int (*fptr)(const void*, const void*);

#define NELEMS(arr) (sizeof(arr) / sizeof(arr[0]))

int numarray[] = {123, 145, 512, 627, 800, 933};

int numeric (const int *p1, const int *p2)
{
   return(*p1 - *p2);
}

#pragma argsused
int lookup(int key)
{
   int  *itemptr;

   /* The cast of (int(*)(const void *,const void*))
      is needed to avoid a type mismatch error at
      compile time */
   itemptr = (int *) bsearch (&key, numarray,
               NELEMS(numarray), sizeof(int),
               (fptr)numeric);
   return (itemptr != NULL);
}

int main(void)
```

```
{
   if (lookup(512))
      printf("512 is in the table.\n");
   else
      printf("512 isn't in the table.\n");

   return 0;
}
```

cabs, cabsl

Funktion

Berechnet den Absolutwert einer komplexen Zahl.

Syntax

```
#include <math.h>
double cabs(struct complex z);
long double cabsl(struct _complexl z);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*cabs*
x		x			*cabsl*

Beschreibung

cabs ist ein Makro, das den absoluten Betrag von *z*, einer Struktur des Typs *complex* berechnet. Dieser Strukturtyp ist in math.h folgendermaßen deklariert:

```
struct complex {
   double x, y;
};
```

wobei *x* der reelle und *y* der imaginäre Bestandteil von *z* ist.

Der Aufruf von *cabs* entspricht einem Aufruf von *sqrt* mit den Quadraten der reellen und imaginären Bestandteile von z, also

```
sqrt(z.x * z.x + z.y * z.y)
```

cabsl ist die **long double** Version und erwartet als Argument eine Struktur des Typs *_complexl*. Sie liefert ein Ergebnis vom Typ **long double** zurück. Der Strukturtyp *_complexl* ist in math.h folgendermaßen definiert:

```
struct _complexl {
   long double x, y;
};
```

Wenn Sie mit C++ arbeiten, ist es besser, den Typ *complex* (definiert in complex.h) und die Funktion *abs* zu verwenden.

Rückgabewert

cabs (oder *cabsl*) liefert den absoluten Wert von *z* als **double** Bei einem Überlauf liefert *cabs* (oder *cabsl*) HUGE_VAL (bzw. _LHUGE_VAL) und setzt *errno* auf

ERANGE Ergebnis außerhalb des Rechenbereichs

Die Fehlerbehandlung für diese Funktion kann über *matherr* und *_matherrl* modifiziert werden.

Siehe auch *abs, complex, fabs, labs, matherr*

Beispiel

```
#include <stdio.h>
#include <math.h>

#ifdef __cplusplus
   #include <complex.h>
#endif

#ifdef __cplusplus /* if C++, use class complex */

   void print_abs(void)
   {
      complex z(1.0, 2.0);
      double  absval;

      absval = abs(z);
      printf("The absolute value of %.2lfi %.2lfj "
             "is %.2lf", real(z), imag(z), absval);
   }

#else  /* below function is for C (and not C++) */

   void print_abs(void)
   {
      struct complex z;
      double absval;

      z.x = 2.0;
      z.y = 1.0;
      absval = cabs(z);

      printf("The absolute value of %.2lfi %.2lfj "
             "is %.2lf", z.x, z.y, absval);
   }

#endif

int main(void)
{
   print_abs();
   return 0;
}
```

calloc

Funktion Reserviert Hauptspeicher.

Syntax
```
#include <stdlib.h>.h
void *calloc(size_t nitems, size_t size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung Über diese Funktion wird der *Heap* verwaltet, in dem Platz für dynamische Variablen belegt und wieder freigegeben werden kann. Viele Datenstrukturen, wie z.B. Bäume und Listen, können nur auf dem Heap untergebracht werden.

In den kleinen Datenmodellen (*tiny*, *small*, *medium*) liegt der Heap zwischen dem Ende des Datensegments und der Spitze des Stacks. Der Heap belegt den gesamten hier zur Verfügung stehenden Speicherbereich bis auf einige Bytes direkt unterhalb der Stack-Spitze, die für Funktionsaufrufe und für DOS reserviert sind.

In den großen Datenmodellen (*compact*, *large*, *huge*) beginnt der Heap oberhalb des Stacks und endet mit der höchsten vom System zur Verfügung gestellten Speicheradresse.

calloc belegt einen Speicherbereich von *nitems* * *size* Bytes, die mit 0 initialisiert werden. Für die Belegung von Speicherblöcken mit mehr als 64 KByte muß die Funktion *farcalloc* verwendet werden.

Rückgabewert *calloc* liefert einen Zeiger auf den neu reservierten Speicherbereich.

Wenn kein Speicherbereich ausreichender Größe zur Verfügung steht oder wenn *nitems* oder *size* den Wert 0 haben, liefert *calloc* NULL zurück.

Siehe auch *farcalloc*, *free*, *malloc*, *realloc*

Beispiel
```
#include <stdio.h>
#include <alloc.h>
#include <string.h>

int main(void)
{
   char *str = NULL;

   /* allocate memory for string */
   str = (char *) calloc(10, sizeof(char));

   /* copy "Hello" into string */
   strcpy(str, "Hello");
```

```
   /* display string */
   printf("String is %s\n", str);

   /* free memory */
   free(str);

   return 0;
}
```

ceil, ceill

Funktion Rundet auf.

Syntax
```
#include <math.h>
double ceil(double x);
long double ceil(long double x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*ceil*
x		x			*ceill*

Beschreibung *ceil* sucht den kleinsten Integerwert größer oder gleich *x*. *ceill* ist die **long double** Version und erwartet ein Argument vom Typ **long double** . Sie liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert Die Funktionen *ceil* und *ceill* liefern ihr ganzzahliges Ergebnis als Wert vom Typ **double** bzw. **long double** zurück.

Siehe auch *floor, fmod*

Beispiel
```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double number = 123.54;
   double down, up;

   down = floor(number);
   up = ceil(number);

   printf("original number     %5.2lf\n", number);
   printf("number rounded down %5.2lf\n", down);
   printf("number rounded up   %5.2lf\n", up);

   return 0;
}
```

_c_exit

Funktion

Führt die gleichen Arbeiten wie *_exit* durch, ohne das laufende Programm zu beenden.

Syntax

```
#include <process.h>
void _c_exit(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

_c_exit führt die gleichen Arbeiten wie *_exit* durch, beendet aber den aufrufenden Prozeß nicht. Interrupt-Vektoren, die durch den Startup-Code modifiziert wurden, werden nicht wiederhergestellt. Weitere Aufräumarbeiten werden nicht durchgeführt.

Rückgabewert

Keiner.

Siehe auch

abort, *atexit*, *_cexit*, *_dos_keep*, *exec...*, *exit*, *_exit*, *signal*, *spawn...*,

Beispiel

```
#include <process.h>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <dos.h>

main()
{
  int fd;
  char c;

  if ((fd = open("_c_exit.c",O_RDONLY)) < 0)
  {
    printf("Unable to open _c_exit.c for reading\n");
    return 1;
  }
  if (read(fd,&c,1) != 1)
    printf("Unable to read from open file handle %d "
           "before _c_exit\n",fd);
  else
    printf("Successfully read from open file "
           "handle %d before _c_exit\n",fd);
  printf("Interrupt zero vector before "
         "_c_exit = %Fp\n", _dos_getvect(0));
  _c_exit();
  if (read(fd,&c,1) != 1)
    printf("Unable to read from open file handle %d "
           "after _c_exit\n",fd);
  else
```

```
    printf("Successfully read from open file "
           "handle %d after _c_exit\n",fd);
  printf("Interrupt zero vector after "
         "_c_exit = %Fp\n",_dos_getvect(0));
  return 0;
}
```

_cexit

Funktion

Führt die gleichen Arbeiten wie *exit* durch, ohne das laufende Programm zu beenden.

Syntax

```
#include <process.h>
void _cexit(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

_cexit führt die gleichen Arbeiten wie *exit* durch, schließt aber keine offenen Dateien und beendet den aufrufenden Prozeß nicht. Sämtliche Dateipuffer werden geschrieben, mit *atexit* festgelegte Exit-Funktionen aufgerufen und durch den Start-up-Code modifizierte Interrupt-Vektoren wiederhergestellt.

Rückgabewert

Keiner.

Siehe auch

abort, *atexit*, *_c_exit*, *_dos_keep*, *exec...*, *exit*, *_exit*, *signal*, *spawn...*

Beispiel

```
#include <process.h>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <dos.h>

main()
{
  int fd;
  char c;

  if ((fd = open("_cexit.c",O_RDONLY)) < 0)
  {
    printf("Unable to open _cexit.c for reading\n");
    return 1;
  }
  if (read(fd,&c,1) != 1)
    printf("Unable to read from open file handle %d "
           "before _cexit\n",fd);
  else
    printf("Successfully read from open file "
```

```
          "handle %d before _cexit\n",fd);
  printf("Interrupt zero vector before "
         "_cexit = %Fp\n",_dos_getvect(0));
  _cexit();
  if (read(fd,&c,1) != 1)
    printf("Unable to read from open file handle %d "
           "after _cexit\n",fd);
  else
    printf("Successfully read from open file "
           "handle %d after _cexit\n",fd);
  printf("Interrupt zero vector after "
         "_cexit = %Fp\n",_dos_getvect(0));
  return 0;
}
```

cgets

Funktion

Liest einen String von der Tastatur.

Syntax

```
#include <conio.h>
char *cgets(char *str);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

cgets liest eine Zeichenfolge von der Tastatur und speichert sie (und die Stringlänge) an der Speicherstelle, auf die **str* zeigt.

cgets liest die Zeichen, bis es eine CR/LF-Kombination antrifft oder bis die maximal zulässige Zeichenzahl gelesen wurde. Vor der Speicherung des Strings wird die eventuell gelesene CR/LF-Kombination durch ein abschließendes Nullzeichen (\0) ersetzt.

Speichern Sie vor dem Aufruf von *cgets* in *str*[*0*] die Maximallänge des zu lesenden Strings. Nach der Rückkehr steht in *str*[*1*] die Anzahl der tatsächlich gelesenen Zeichen. Die gelesenen Zeichen werden ab *str*[*2*] gespeichert und enden mit einem Nullzeichen. *str* muß daher mindestens *str*[*0*] + 2 Bytes lang sein.

Rückgabewert

cgets liefert einen Zeiger auf *str*[*2*] zurück, d.h. auf die Adresse des ersten eingelesenen Zeichens.

Siehe auch

cputs, *fgets*, *getch*, *getche*, *gets*

Beispiel

```
#include <stdio.h>
#include <conio.h>

main()
{
```

```
    char buffer[83];
    char *p;

    /* There's space for 80 characters */
    /* plus the NULL terminator */
    buffer[0] = 81;

    printf("Input some chars:");
    p = cgets(buffer);
    printf("\ncgets read %d characters: \"%s\"\n",
           buffer[1], p);
    printf("The returned pointer is %p, buffer[0] "
           "is at %p\n", p, &buffer);

    /* Leave room for 5 characters */
    /* plus the NULL terminator */
    buffer[0] = 6;

    printf("Input some chars:");
    p = cgets(buffer);
    printf("\ncgets read %d characters: \"%s\"\n",
           buffer[1], p);
    printf("The returned pointer is %p, buffer[0] "
           "is at %p\n", p, &buffer);
    return 0;
}
```

_chain_intr

Funktion: Übergibt die Steuerung an einen anderen Interupt-Handler.

Syntax:

```
#include <dos.h>
void _chain_intr(void(interrupt far *newhandler)());
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung: *_chain_intr* übergibt die Steuerung vom aktuell ausgeführten Interrupt-Handler an den Interrupt-Handler mit der Adresse *newhandler*. Das aktuelle Register wird dabei *nicht* an den neuen Handler übergeben. Der neue Interrupt-Handler übernimmt stattdessen die Register, die vom alten Handler auf dem Stack plaziert und dort möglicherweise modifiziert wurden. Der neue Interrupt-Handler kann zurückkehren, als wäre er der alte Handler. Dieser wird nicht wieder aktiviert.

_chain_intr kann nur von C-Interrupt-Funktionen aufgerufen werden. Die Funktion ist hilfreich beim Schreiben eines speicherresidenten Programms (TSR), das sich selbst in die

Kette der Interrupt-Handler (z.B. dem Keyboard-Interrupt) einfügen muß.

Rückgabewert *_chain_intr* liefert keinen Wert zurück.

Siehe auch *_dos_getvect, _dos_setvect, _dos_keep*

Beispiel

```
#include <dos.h>
#include <stdio.h>
#include <process.h>

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

typedef void interrupt (*fptr)(__CPPARGS);

static void mesg(char *s)
{
  while (*s)
    bdos(2,*s++,0);
}

#pragma argsused
void interrupt handler2(unsigned bp, unsigned di)
{
  _enable();
  mesg("In handler 2.\r\n");
  if (di == 1)
    mesg("DI is 1\r\n");
  else
    mesg("DI is not 1\r\n");
  di++;
}

#pragma argsused
void interrupt handler1(unsigned bp, unsigned di)
{
  _enable();
  mesg("In handler 1.\r\n");
  if (di == 0)
    mesg("DI is 0\r\n");
  else
    mesg("DI is not 0\r\n");
  di++;
  mesg("Chaining to handler 2.\r\n");
  _chain_intr(handler2);
}
```

```
void main()

{
  _dos_setvect(128,(fptr) handler1);
  printf("About to generate interrupt 128\n");
  _DI = 0;
  geninterrupt(128);
  printf("DI was 0 before interrupt, is now 0x%x\n",
         _DI);
  exit(0);
}
```

chdir

Funktion — Wechselt das aktuelle Verzeichnis.

Syntax

```
#include <dir.h>
int chdir(const char *path);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *chdir* setzt das durch *path* bezeichnete Verzeichnis als aktuelles Arbeitsverzeichnis. Der Parameter *path* muß ein existierendes Verzeichnis bezeichnen. Er kann auch eine Laufwerksangabe enthalten.

In einem solchen Fall:

```
chdir ("a:\\BC")
```

wird jedoch nur das aktuelle Verzeichnis dieses Laufwerks gewechselt, nicht das aktive Laufwerk selbst.

Rückgabewert — Wenn das angegebene Verzeichnis gesetzt werden konnte, liefert *chdir* den Wert 0; ansonsten wird -1 zurückgeliefert und die globale Variable *errno* bekommt den Wert:

ENOENT Suchweg/Datei nicht gefunden

Siehe auch — *getcurdir, getcdw, getdisk, mkdir, rmdir, setdisk, system*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>

char old_dir[MAXDIR];
char new_dir[MAXDIR];

int main(void)
{
   if (getcurdir(0, old_dir))
```

```
    {
       perror("getcurdir()");
       exit(1);
    }
    printf("Current directory is: \\%s\n", old_dir);

    if (chdir("\\"))
    {
       perror("chdir()");
       exit(1);
    }

    if (getcurdir(0, new_dir))
    {
       perror("getcurdir()");
       exit(1);
    }
    printf("Current directory is now: \\%s\n",
           new_dir);

    printf("\nChanging back to original directory: "
           "\\%s\n", old_dir);
    if (chdir(old_dir))
    {
       perror("chdir()");
       exit(1);
    }

    return 0;
}
```

_chdrive

Funktion — Setzt das aktuelle Laufwerk.

Syntax

```
#include <direct.h>
int _chdrive(int drive);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *_chdrive* macht das durch *drive* bezeichnete Laufwerk (1 = A, 2 = B, 3 = C, usw.) zum aktuellen Laufwerk.

Rückgabewert — Wenn das angegebene Laufwerk gesetzt werden konnte, liefert *_chdrive* den Wert 0; ansonsten wird -1 zurückgeliefert.

Siehe auch — *_dos_setdrive*, *_getdrive*

Beispiel

```
#include <stdio.h>
#include <direct.h>

int main(void)
{
  if (_chdrive(3) == 0)
    printf("Successfully changed to drive C:\n");
  else
    printf("Cannot change to drive C:\n");
  return 0;
}
```

_chmod

Funktion Liest bzw. ändert die DOS-Dateiattribute.

Syntax

```
#include <dos.h>
#include <io.h>
int _chmod(const char *path, int func [,int attrib]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Über *_chmod* kann die Zugriffsberechtigung der durch *path* angegebenen Datei entweder gelesen (*func* = 0) oder auf den Wert von *attrib* gesetzt werden (*func* = 1). Für den Parameter *attrib* sind die folgenden Konstanten in dos.h definiert:

FA_RDONLY	Nur Lesen
FA_HIDDEN	Verborgen
FA_SYSTEM	Systemdatei
FA_LABEL	Volume-Label (Datenträgername)
FA_DIREC	Verzeichnis
FA_ARCH	Archiv

Rückgabewert Bei fehlerfreier Ausführung liefert *_chmod* das gelesene bzw. neu gesetzte Attribut.

Im Fall eines Fehlers ist der Rückgabewert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

ENOENT	Suchweg/Datei nicht gefunden
EACCES	Zugriff nicht erlaubt

Siehe auch *chmod*, *_creat*

Beispiel

```
#include <errno.h>
#include <stdio.h>
#include <dos.h>
#include <io.h>

int get_file_attrib(char *filename);
```

```
int main(void)
{
   char filename[128];
   int attrib;

   printf("Enter a filename:");
   scanf("%s", filename);
   attrib = get_file_attrib(filename);
   if (attrib == -1)
      switch(errno)
      {
         case ENOENT : printf("Path or file not "
                              "found.\n");
                       break;
         case EACCES : printf("Permission denied.\n");
                       break;
         default:      printf("Error number: %d",
                             errno);
                       break;
      }
   else
      {
         if (attrib & FA_RDONLY)
            printf("%s is read-only.\n", filename);

         if (attrib & FA_HIDDEN)
            printf("%s is hidden.\n", filename);

         if (attrib & FA_SYSTEM)
            printf("%s is a system file.\n",
                   filename);

         if (attrib & FA_LABEL)
            printf("%s is a volume label.\n",
                   filename);

         if (attrib & FA_DIREC)
            printf("%s is a directory.\n", filename);

         if (attrib & FA_ARCH)
            printf("%s is an archive file.\n",
                   filename);
      }
   return 0;
}

/* returns the attributes of a DOS file */
int get_file_attrib(char *filename)
```

```
{
   return(_chmod(filename, 0));
}
```

chmod

Funktion

Ändert die Zugriffsmöglichkeiten auf eine Datei.

Syntax

```
#include <sys\stat.h>
int chmod(const char *path, int amode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

chmod setzt die Zugriffsberechtigung der durch *path* angegebenen Datei auf den durch *amode* angegebenen Wert. *path* zeigt auf einen String, wobei **path* das erste Zeichen dieses Strings ist.

amode kann eine (oder beide) der in sys\stat.h definierten symbolischen Kostanten S_IWRITE und S_IREAD enthalten.

Wert von *amode*	Zugriffsberechtigung
S_IWRITE	Schreiben erlaubt
S_IREAD	Lesen erlaubt
S_IREAD | S_IWRITE	Lesen und Schreiben erlaubt

Rückgabewert

Bei fehlerfreier Ausführung liefert *chmod* den Wert 0.

Im Fall eines Fehlers ist der Rückgabewert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

ENOENT	Suchweg/Datei nicht gefunden
EACCES	Zugriff nicht erlaubt

Siehe auch

access, *_chmod*, *fstat*, *open*, *sopen*, *stat*

Beispiel

```
/* NEW chmod() example: */

#include <errno.h>
#include <stdio.h>
#include <io.h>
#include <process.h>
#include <sys\stat.h>

void main(void)
{
  char filename[64];
  struct stat stbuf;
  int amode;

  printf("Enter name of file: ");
```

```
  scanf("%s", filename);
  if (stat(filename, &stbuf) != 0)
  {
    perror("Unable to get file information");
    exit(1);
  }
  if (stbuf.st_mode & S_IWRITE)
  {
    printf("Changing to read-only\n");
    amode = S_IREAD;
  }
  else
  {
    printf("Changing to read-write\n");
    amode = S_IREAD|S_IWRITE;
  }
  if (chmod(filename, amode) != 0)
  {
    perror("Unable to change file mode");
    exit(1);
  }
  exit(0);
}
```

chsize

Funktion — Ändert die Größe einer Datei.

Syntax

```
#include <io.h>
int chsize(int handle, long size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *chsize* ändert die Größe der mit *handle* verbundenen Datei. Wenn diese Datei mehr als *size* Bytes umfaßt, wird sie entsprechend verkürzt (d.h. alle Zeichen hinter dem neuen Dateiende-Flag werden gelöscht). Andernfalls wird sie mit NULL-Zeichen ('\0') aufgefüllt.

Die Datei muß in beiden Fällen in einem Modus geöffnet worden sein, der Schreibaktionen erlaubt.

Rückgabewert — Bei fehlerfreier Ausführung liefert *chsize* den Wert 0. Ansonsten ist der Rückgabewert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

EACCESS	Zugriff verweigert
EBADF	Handle ungültig
ENOSPC	UNIX – nicht DOS

Siehe auch *close, _creat, creat, open*

Beispiel

```
#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* create text file containing 10 bytes */
   handle = open("DUMMY.FIL", O_CREAT);
   write(handle, buf, strlen(buf));

   /* truncate the file to 5 bytes in size */
   chsize(handle, 5);

   /* close the file */
   close(handle);
   return 0;
}
```

circle

Funktion Zeichnet einen Kreis mit dem angegebenen Radius um den Mittelpunkt (*x*,*y*).

Syntax

```
#include <graphics.h>
void far circle(int x, int y, int radius);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *circle* zeichnet einen Kreis mit dem durch *radius* angegebenen Radius um den Mittelpunkt (*x*,*y*). Es wird die aktuelle Zeichenfarbe verwendet.

Hinweis: Der Parameter *linestyle* von *setlinestyle* für die Linienart wirkt sich auf *arc*, *circle*, *ellipse*, *fillellipse*, *sector* und *pieslice* nicht aus; der Parameter *thickness* für die Liniendicke wird dagegen verwendet.

Wenn Kreise nicht ganz rund ausehen sollten, kann dies durch den Aufruf von *setaspectratio* korrigiert werden.

Rückgabewert Keiner.

Siehe auch *arc*, *ellipse*, *fillellipse*, *getaspectratio*, *sector*, *setaspectratio*, *setlinestyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int radius = 100;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* draw the circle */
   circle(midx, midy, radius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

_clear87

Funktion Löscht das Status-WORD für Gleitkommabearbeitung.

Syntax

```
#include <float.h>
unsigned int _clear87(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_clear87* löscht das Status-WORD für Gleitkommabearbeitung, dessen Inhalt sich aus dem Statuswort des 80x87 und einigen Bits zusammensetzt, die von der Routine zur Behandlung von Ausnahmezuständen des 80x87 entdeckt werden.

Rückgabewert Es wird der vorherige Inhalt des Statuswortes zurückgeliefert, dessen einzelne Bits in float.h dokumentiert sind.

Siehe auch *_control87*, *_fpreset*, *_status87*

Beispiel

```
#include <stdio.h>
#include <float.h>

int main(void)
{
   float x;
   double y = 1.5e-100;

   printf("\nStatus 87 before error: %X\n",
          _status87());

   x = y; /* create underflow and precision loss */
   printf("Status 87 after  error: %X\n",
          _status87());

   _clear87();
   printf("Status 87 after  clear: %X\n",
          _status87());

   y = x;

   return 0;
}
```

cleardevice

Funktion Löscht den Grafikbildschirm.

Syntax

```
#include <graphics.h>
void far cleardevice(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *cleardevice* löscht den gesamten Grafikbildschirm, füllt ihn also mit der aktuellen Hintergrundfarbe und setzt die aktuelle Position (CP) auf den Punkt (0,0) in die obere linke Ecke des Grafikbildschirms.

Rückgabewert Keiner.

Siehe auch *clearviewport*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* for centering screen messages */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* output a message to the screen */
   outtextxy(midx, midy,
             "press any key to clear the screen:");

   /* wait for a key */
   getch();

   /* clear the screen */
   cleardevice();

   /* output another message */
   outtextxy(midx, midy, "press any key to quit:");

   /* clean up */
   getch();
```

```
   closegraph();
   return 0;
}
```

clearerr

Funktion — Setzt das Fehler- und das Dateiende-Flag für einen bestimmten Stream zurück.

Syntax

```
#include <stdio.h>
void clearerr(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *clearerr* setzt das Fehler-Flag und das Dateiende-Flag des durch *stream* bezeichneten Stream auf 0 zurück. Das Fehler-Flag einer Datei bleibt nach einem Fehler solange gesetzt, bis entweder *clearerr* oder *rewind* aufgerufen oder die Datei geschlossen wird.

Das Dateiende-Flag wird bei jeder Eingabeoperation in die Datei zurückgesetzt.

Rückgabewert — Keiner.

Siehe auch — *eof*, *feof*, *ferror*, *perror*, *rewind*

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *fp;
   char ch;

   /* open a file for writing */
   fp = fopen("DUMMY.FIL", "w");

  /* force an error condition by attempting to read */
   ch = fgetc(fp);
   printf("%c\n",ch);

   if (ferror(fp))
   {
      /* display an error message */
      printf("Error reading from DUMMY.FIL\n");

      /* reset the error and EOF indicators */
      clearerr(fp);
   }
```

```
   fclose(fp);
   return 0;
}
```

clearviewport

Funktion Löscht den Inhalt des aktuellen Zeichenfensters.

Syntax
```
#include <graphics.h>
void far clearviewport(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *clearviewport* löscht den Inhalt des aktuellen Zeichenfensters und setzt die aktuelle Position (CP) auf den Ursprung (0,0), relativ zum Zeichenfenster.

Rückgabewert Keiner.

Siehe auch *cleardevice*, *getviewsettings*, *setviewport*

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define CLIP_ON 1 /* activates clipping in viewport */

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int ht;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
```

```
    setcolor(getmaxcolor());
    ht = textheight("W");

    /* message in default full-screen viewport */
    outtextxy(0, 0, "* <-- (0, 0) in "
              "default viewport");

    /* create a smaller viewport */
    setviewport(50, 50, getmaxx()-50, getmaxy()-50,
                CLIP_ON);

    /* display some messages */
    outtextxy(0, 0, "* <-- (0, 0) in "
              "smaller viewport");
    outtextxy(0, 2*ht, "Press any key "
              "to clear viewport:");

    /* wait for a key */
    getch();

    /* clear the viewport */
    clearviewport();

    /* output another message */
    outtextxy(0, 0, "Press any key to quit:");

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

clock

Funktion

Berechnet die seit dem Programmaufruf verbrauchte »CPU-Zeit«.

Syntax

```
#include <time.h>
clock_t clock(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung

clock kann verwendet werden, um die zwischen zwei Ereignissen vergangene Zeit zu bestimmen.

Um die Zeit in Sekunden zu berechnen, kann die von *clock* gelieferte Zeit durch die Konstante CLK_TCK geteilt werden.

Rückgabewert *clock* liefert die seit dem Programmaufruf bis jetzt verbrauchte »CPU-Zeit« zurück. Falls die Prozessorzeit nicht verfügbar ist oder deren Wert nicht dargestellt werden kann, liefert die Funktion -1 zurück.

Siehe auch *time*

Beispiel

```
#include <time.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   clock_t start, end;
   start = clock();

   delay(2000);

   end = clock();
   printf("The time was: %f\n",
          (end - start) / CLK_TCK);

   return 0;
}
```

_close, close

Funktion Schließt eine Datei.

Syntax

```
#include <io.h>
int _close(int handle);
int close(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*_close*
x	x	x			*close*

Beschreibung *_close* und *close* schließen die durch *handle* bezeichnete Datei.

Ein Handle wird durch einen Aufruf von *_creat*, *creat*, *creatnew*, *creattemp*, *dup*, *dup2*, *open* oder *_open* einer Datei zugeordnet.

Hinweis: Diese Funktionen schreiben keine »eigene« Daten (wie zum Beispiel ein ^Z als Dateiende-Zeichen). Falls Sie eine Datei mit ^Z abschließen wollen, müssen Sie dieses Zeichen explizit ausgeben.

Rückgabewert *_close* und *close* liefern den Wert 0 zurück, wenn die Operation fehlerfrei ausgeführt werden konnte, ansonsten -1.

handle muß ein gültiges Handle sein, dem eine gültige, offene Datei zugeordnet ist. Falls dem nicht so ist, bekommt die globale Variable *errno* den Wert:

EBADF Ungültiges Handle

Siehe auch *chsize*, *_creat*, *creat*new, *dup*, *fclose*, *open*, *sopen*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

main()
{
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  handle = open("NEW.FIL", O_CREAT);
  if (handle > -1)
  {
    write(handle, buf, strlen(buf));

    close(handle);              /* close the file */
  }
  else
  {
    printf("Error opening file\n");
  }
  return 0;
}
```

closedir

Funktion Schließt einen Verzeichnis-Stream.

Syntax

```
#include <dirent.h>
void closedir(DIR *dirp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung Unter UNIX ist *closedir* auf POSIX-Systemen verfügbar. Die Funktion schließt den Verzeichnis-Stream *dirp*, der vorher durch einen Aufruf von *opendir* geöffnet wurde.

Nachdem der Stream geschlossen wurde, zeigt **dirp* nicht mehr auf einen gültigen Verzeichnis-Stream.

Rückgabewert Bei fehlerfreier Ausführung liefert *closedir* den Wert 0. Ansonsten ist der Rückgabewert -1 und die globale Variable *errno* wird auf den folgenden Wert gesetzt:

EBADF	Der Parameter *dirp* zeigt nicht auf einen gültigen geöffneten Verzeichnis-Stream.

Siehe auch *opendir, readdir, rewinddir*

Beispiel Vgl. das Beispiel zu *opendir*.

closegraph

Funktion Beendet den Grafikmodus.

Syntax
```
#include <graphics.h>
void far closegraph(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *closegraph* gibt den vom Grafikpaket für Treiber, Fonts und interne Puffer belegten Speicherplatz über einen Aufruf von *_graphfreemem* wieder frei und schaltet in den Modus zurück, der vor dem Aufruf von *initgraph* gesetzt war.

Rückgabewert Keiner.

Siehe auch *initgraph, setgraphbufsize*

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
```

```
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "Press a key to close "
             "the graphics system:");

   /* wait for a key */
   getch();

   /* closes down the graphics system */
   closegraph();

   printf("We're now back in text mode.\n");
   printf("Press any key to halt:");
   getch();
   return 0;
}
```

clreol

Funktion

Löscht alle Zeichen von der aktuellen Cursorposition bis zum Zeilenende.

Syntax

```
#include <conio.h>
void clreol(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

clreol löscht innerhalb des aktuellen Textfensters die Spalten rechts vom Cursor bis zum Ende der Zeile. Die Position des Cursors bleibt unverändert.

Rückgabewert

Keiner.

Siehe auch

Beispiel

```
#include <conio.h>

int main(void)

{
```

```
   clrscr();
   cprintf("The function CLREOL clears all "
           "characters from the\r\n");
   cprintf("cursor position to the end of the "
           "line within the\r\n");
   cprintf("current text window, without moving "
           "the cursor.\r\n");
   cprintf("Press any key to continue . . .");
   gotoxy(14, 4);
   getch();

   clreol();
   getch();

   return 0;
}
```

clrscr

Funktion — Löscht den Inhalt des aktuellen Textfensters.

Syntax

```
#include <conio.h>
void clrscr(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *clrscr* löscht den gesamten Inhalt des aktuellen Textfensters und plaziert den Cursor in die obere linke Ecke des Textfensters, d.h. auf die Koordinaten (1,1).

Rückgabewert — Keiner.

Siehe auch — *clreol, delline, window*

Beispiel

```
#include <conio.h>

int main(void)
{
   int i;

   clrscr();
   for (i = 0; i < 20; i++)
      cprintf("%d\r\n", i);
   cprintf("\r\nPress any key to clear screen");
   getch();

   clrscr();
   cprintf("The screen has been cleared!");
```

```
   getch();

   return 0;
}
```

complex

Funktion Erzeugt eine komplexe Zahl.

Syntax
```
#include <complex.h>
complex complex(double real, double imag);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung *complex* erzeugt aus gegebenen realen und imaginären Anteilen eine komplexe Zahl. Wenn *imag* nicht übergeben wird, wird der imaginäre Anteil auf 0 gesetzt.

complex ist der Konstruktor für die C++ Klasse *complex* und – zusammen mit anderen Funktionen – in complex.h definiert. Einige dieser Funktionen sind überladene Versionen von in math.h deklarierten C-Funktionen. Für die *complex*-Versionen ist C++ erforderlich.

Wenn Sie nicht in C++, sondern in C programmieren, stehen Ihnen nur die Konstrukte *struct complex* und *cabs* zur Verfügung. Beide sind in math.h deklariert und berechnen den absoluten Wert einer komplexen Zahl.

In complex.h werden auch die Operatoren +, -, *, /, +=, -=, *=, /=, =, == und != überladen. Mit diesen Operatoren ist komplexe Arithmetik im üblichen Sinne möglich. Komplexe Zahlen können in Ausdrücken beliebig mit **int**, **double** und anderen numerischen Typen gemischt werden. Die Operatoren << und >> sind für die Stream-Ein- und Ausgabe von komplexen Zahlen überladen, genauso wie für die anderen Datenypen in iostream.h.

Rückgabewert *complex* liefert die komplexe Zahl mit dem gegebenen realen und imaginären Anteil.

Siehe auch *abs*, *acos*, *arg*, *asin*, *atan*, *atan2*, *conj*, *cos*, *cosh*, *imag*, *log*, *log10*, *norm*, *polar*, *pow*, *real*, *sin*, *sinh*, *sqrt*, *tan*, *tanh*

Beispiel
```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
```

```
   cout << "z = "<< z << "\n";
   cout << " and imaginary real part = "
        << imag(z) << "\n";
   cout << "z has complex conjugate =  "
        << conj(z) << " \n";
   return 0;
}
```

conj

Funktion Berechnet zu einer komplexen Zahl die konjugiert komplexe Zahl.

Syntax
```
#include <complex.h>
complex conj(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung *conj*(z) ist identisch mit *complex*(*real*(z), *-imag*(z)).

Rückgabewert *conj* liefert die konjugiert komplexe Zahl zurück.

Siehe auch *complex, imag, real*

Beispiel
```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = "<< z << "\n";
   cout << " and imaginary real part = "
        << imag(z) << "\n";
   cout << "z has complex conjugate =  "
        << conj(z) << " \n";
   return 0;
}
```

_control87

Funktion Liest und setzt das Steuer-WORD für Gleitkommaoperationen.

Syntax
```
#include <float.h>
unsigned int _control87(unsigned int newcw,
                        unsigned int mask);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

_control87 liefert oder ändert das Steuer-WORD für Gleitkommaoperationen. Über die Bits dieses WORD – ein **unsigned int**-Wert – werden insbesondere die Modi für Genauigkeit, Rundungsrichtung und die Behandlung unendlicher Werte in den Gleitkommaroutinen festgelegt. Durch Änderung dieser Modi können Ausnahmesituationen bei Gleitkommarechnungen unterschiedlich behandelt werden.

Die einzelnen Bits von *mask* bestimmen, welche Bits des Steuer-WORD verändert werden, die entsprechenden Bits von *newcw* definieren die neuen Werte. Wenn ein Bit von *mask* auf 1 gesetzt ist, dann wird das entsprechende Bit des Steuer-WORD auf den Wert gesetzt, den das korrespondierende Bit von *newcw* hat. Ein Beispiel:

Ursprüngliches Steuer-WORD:	0100	0011	0110	0011
mask	1000	0001	0100	1111
newcw	1110	1001	0000	0101
Geänderte Bits	1*xxx*	*xxx*1	*x*0*xx*	0101

Wenn *mask* 0 ist, liefert *_control87* das Steuer-WORD zurück, ohne es zu ändern.

Das DENORMAL-Bit wird durch *_control87* nicht verändert, da C++ diese Ausnahmezustände benutzt.

Rückgabewert

_control87 liefert das neue Steuer-WORD zurück, das als Bitmuster interpretiert werden muß. Eine vollständige Definition aller zurückgelieferten Bits finden Sie in float.h.

Siehe auch

_clear87, *_fpreset*, *signal*, *_status87*

coreleft

Funktion

Ermittelt, wieviele Bytes des Hauptspeichers noch unbelegt und damit verfügbar sind.

Syntax

```
#include <alloc.h>
unsigned coreleft(void);      /* tiny, small, medium */
unsigned long coreleft(void);/* compact, large, huge */
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

coreleft liefert ein Maß für die Größe des unbenutzten Speicherplatzes. Das zurückgelieferte Maß hängt vom verwendeten Speichermodell ab.

Rückgabewert *coreleft* liefert in den kleinen Datenmodellen die Größe des ungenutzen Speichers zwischen der Spitze des Heap und dem Stack zurück. In den großen Datenmodellen liefert *coreleft* die Größe zwischen der höchsten verfügbaren Speicheradresse und der Endadresse des höchsten belegten Blocks zurück.

Siehe auch *allocmem, brk, farcoreleft, malloc*

Beispiel

```
#include <stdio.h>
#include <alloc.h>

int main(void)
{
  printf("The difference between the highest "
         "allocated block and\n");
  printf("the top of the heap is: %lu bytes\n", "
         (unsigned long) coreleft());
  return 0;KEEPFOLLOWING
}
```

cos, cosl

Funktion Berechnet den Cosinus des Arguments.

Syntax

reelle Version:

```
#include <math.h>
double cos(double x);
long double cos(long double x);
```

komplexe Version:

```
#include <complex.h>
complex cos(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*cosl*
x	x	x	x		*reelles cos*
x		x		x	*komplexes cos*

Beschreibung *cos* erwartet einen Winkel in der Einheit *rad* als Argument und liefert den dazugehörigen Cosinus zurück.

cosl ist die **long double** Version und erwartet ein Argument vom Typ **long double** . Sie liefert ein Ergebnis vom Typ **long double** zurück.

Die komplexe Version von *cos* ist wie folgt definiert:

$$cos(z) = (exp\ (i * z) + exp\ (-i * z)) / 2$$

Rückgabewert Für ein reelles Argument liefert *cos* Werte im Bereich von -1 bis +1 zurück.

Die Fehlerbehandlung für *cos* und *cosl* kann mit *matherr* bzw. *_matherrl* modifiziert werden.

Siehe auch *acos, asin, atan, atan2, complex, matherr, sin, tan*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;

   result = cos(x);
   printf("The cosine of %lf is %lf\n", x, result);
   return 0;
}
```

cosh, coshl

Funktion Berechnet den hyperbolischen Cosinus eines Arguments.

Syntax

reelle Version:
```
#include <math.h>
double cosh(double x);
long double coshl(long double x);
```

komplexe Version:
```
#include <complex.h>
complex cosh(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*coshl*
x	x	x	x		*reelles cosh*
x		x		x	*komplexes cosh*

Beschreibung *cosh* berechnet den Cosinus hyperbolicus $(e^{x} + e^{-x}) / 2$.
coshl ist die **long double** Version und erwartet ein Argument vom Typ **long double** . Sie liefert ein Ergebnis vom Typ **long double** zurück.

Die komplexe Version von *cosh* ist wie folgt definiert:

$cosh(z) = (exp\ (z) + exp\ (-z)) / 2$

Rückgabewert Bei fehlerfreier Ausführung liefert *cosh* den Cosinus hyperbolicus des Arguments.

Bei Überläufen liefert *cosh* den Wert HUGE_VAL und *coshl* den Wert _LHUGE_VAL (jeweils mit korrektem Vorzeichen) zurück, und *errno* wird auf ERANGE (Ergebnis außerhalb des Rechenbereichs) gesetzt.

Die Behandlung von Fehlern kann für diese Funktionen über die Routinen *matherr* bzw. *_matherrl* modifiziert werden.

Siehe auch *acos, asin, atan, atan2, complex, cos, matherr, sin, sinh, tan, tanh*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
```

```
{
   double result;
   double x = 0.5;

   result = cosh(x);
   printf("The hyperbolic cosine of %lf is %lf\n",
          x, result);
   return 0;
}
```

country

Funktion

Liefert Informationen über nationale Formate für Uhrzeit, Datum und numerische Werte.

Syntax

```
#include <dos.h>
struct COUNTRY *country(int xcode, struct COUNTRY *cp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

country setzt nationale Formate für numerische Werte, Trennzeichen, Uhrzeit und Datum, wobei die tatsächlich gesetzten Werte von der verwendeten DOS-Version abhängig sind.

Wenn *cp* den Wert -1 hat, wird das länderspezifische Format auf den durch *xcode* angegebenen Wert gesetzt, der in diesem Fall ungleich Null sein muß.

Ansonsten setzt *country* die Struktur, auf die *cp* zeigt: wenn *xcode* den Wert 0 hat, entsprechen die Werte den aktuell von DOS verwendeten landesspezifischen Einstellungen, wenn *xcode* ungleich Null ist, werden die Werte des entsprechenden Landes ermittelt (unabhängig davon, welche Werte DOS aktuell benutzt).

Die Struktur COUNTRY ist in dos.h wie folgt definiert:

```
struct COUNTRY {
   int  co_date;        /* Datumsformat */
   char co_curr[5];     /* Währungssymbol */
   char co_thsep[2];    /* Trennzeichen für 1000er */
   char co_desep[2];    /* Dezimaltrennzeichen */
   char co_dtsep[2];    /* Datumstrennzeichen */
   char co_tmsep[2];    /* Uhrzeittrennzeichen */
   char  co_currstyle;  /* Währungsformat */
   char  co_digits;     /* Signifikante Ziffern */
   char  co_time;       /* Uhrzeit: 12 / 24 Stunden */
   long  co_case;       /* Groß-/Kleinschreibung */
   char  co_dasep[2];   /* Datentrennzeichen */
   char  co_fill[10];   /* Füllzeichen */
};
```

Der Integer *co_date* kann drei Werte annehmen:

0 USA-Format: Monat, Tag, Jahr

1 europäisches Format: Tag, Monat, Jahr

2 japanisches Format: Jahr, Monat, Tag

Das Element *co_currstyle* legt fest, in welchem Format Geldbeträge ausgegeben werden:

0 das Währungssymbol steht ohne Leerzeichen vor dem Betrag ("DM1000");

1 das Währungssymbol steht ohne Leerzeichen hinter dem Betrag ("1000DM");

2 wie 0, nur mit einem Leerzeichen zwischen Währungssymbol und Betrag:

3 wie 1, nur mit einem Leerzeichen zwischen Betrag und Währungssymbol.

Rückgabewert Bei fehlerfreier Ausführung liefert *country* das Zeiger-Argument *cp* zurück, andernfalls NULL.

Beispiel

```
#include <dos.h>
#include <stdio.h>

#define USA 0

int main(void)
{
   struct COUNTRY country_info;

   country(USA, &country_info);
   printf("The currency symbol for the USA is: %s\n",
           country_info.co_curr);
   return 0;
}
```

cprintf

Funktion Gibt formatiert auf den Bildschirm aus.

Syntax

```
#include <conio.h>
int cprintf(const char *format[, argument, ...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *cprintf* »druckt« in formatierter Form und erwartet zumindest einen Formatstring (Parameter *format*), über den die Anzahl

der Ausgaben und ihr Format festgelegt wird. Die Ausgabe erfolgt direkt in das aktuelle Textfenster des Bildschirms. Die Anzahl der übergebenen Argumente und Formatbezeichner muß übereinstimmen.

Abhängig vom Wert der globalen Variablen *directvideo* wird der String entweder direkt in den Bildschirmspeicher geschrieben oder über den Umweg eines BIOS-Aufrufs ausgegeben.

Anders als bei *fprintf* und *printf* wird keine »Übersetzung« des Zeichens *\n* (Zeilenvorschub) in *\r\n* (Wagenrücklauf/Zeilenvorschub) vorgenommen.

Hinweis: Die Beschreibung der Funktion *printf* enthält eine genaue Erläuterung der Verwendung von Formatbezeichnern.

Rückgabewert *cprintf* liefert die Anzahl der ausgegebenen Zeichen zurück.

Siehe auch *directvideo* (globale Variable), *fprintf*, *printf*, *putch*, *sprintf*, *vprintf*

Beispiel

```
#include <conio.h>

int main(void)
{
   /* clear the screen */
   clrscr();

   /* create a text window */
   window(10, 10, 80, 25);

   /* output some text in the window */
   cprintf("Hello world\r\n");

   /* wait for a key */
   getch();
   return 0;
}
```

cputs

Funktion Gibt einen String auf den Bildschirm aus.

Syntax

```
#include <conio.h>
int cputs(const char *str);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *cputs* schreibt den nullterminierten String *str* auf den Bildschirm in das aktuelle Textfenster. Diese Funktion erzeugt *keinen* automatischen Zeilenvorschub am Ende der Ausgabe.

Abhängig vom Wert der globalen Variablen *directvideo* wird der String entweder direkt in den Bildschirmspeicher geschrieben oder über den Umweg eines BIOS-Aufrufs ausgegeben.

Anders als bei *puts* wird keine »Übersetzung« des Zeichens *\n* (Zeilenvorschub) in *\r\n* (Wagenrücklauf/Zeilenvorschub) vorgenommen.

Rückgabewert *cputs* liefert das zuletzt geschriebene Zeichen zurück.

Siehe auch *cgets*, *directvideo* (globale Variable), *fputs*, *putch*, *puts*

Beispiel

```
#include <conio.h>

int main(void)
{
   /* clear the screen */
   clrscr();

   /* create a text window */
   window(10, 10, 80, 25);

   /* output some text in the window */
   cputs("This is within the window\r\n");

   /* wait for a key */
   getch();
   return 0;
}
```

_creat, _dos_creat

Funktion Erzeugt eine neue Datei oder überschreibt eine bereits vorhandene Datei.

Syntax

```
#include <dos.h>
int _creat(const char *path, int attrib);
unsigned _dos_creat(const char *path, int attrib
                    int *handlep);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*_creat*
x		x			*_dos_creat*

Beschreibung *_creat* und *_dos_creat* öffnen die durch *path* bezeichnete Datei grundsätzlich im binären Modus. Nach einer fehlerfreien Eröffnung bzw. Erzeugung zeigt der Dateizeiger auf das erste Byte der Datei. *_dos_creat* speichert das Datei-Handle an der durch *handlep* bezeichneten Position. Die Datei wird für Lese- und Schreibzugriffe geöffnet.

Wenn die Datei bereits existiert, wird sie auf Null Bytes Länge verkürzt (d.h. der vorherige Inhalt wird gelöscht).

Für *_creat* ist der Parameter *attrib* eine ODER-Kombination aus einer oder mehreren der folgenden in dos.h definierten Konstanten:

FA_RDONLY	Nur-Lesen-Attribut
FA_HIDDEN	Verborgen
FA_SYSTEM	Systemdatei

Für *_dos_creat* ist der Parameter *attrib* eine ODER-Kombination aus einer oder mehreren der folgenden in dos.h definierten Konstanten:

_A_NORMAL	Normale Datei
_A_RDONLY	Nur-Lesen-Datei
_A_HIDDEN	Verborgene Datei
_A_SYSTEM	Systemdatei

Rückgabewert Bei fehlerfreier Ausführung liefert *_creat* das neue Datei-Handle (einen positiven Integerwert) zurück. Ansonsten ist der Rückgabewert -1.

_dos_creat liefert bei fehlerfreier Ausführung 0, andernfalls wird der DOS-Fehlercode zurückgeliefert.

Im Fall eines Fehlers wird die globale Variable *errno* auf einen der folgenden Werte gesetzt (dies gilt für *_creat* und *_dos_creat*):

ENOENT	Suchweg/Datei nicht gefunden
EMFILE	Zu viele offene Dateien
EACCES	Zugriff nicht erlaubt

Siehe auch *_chmod*, *chsize*, *_close*, *close*, *creat*, *creatnew*, *creattemp*

Beispiel

```
#include <dos.h>
#include <string.h>
#include <stdio.h>
#include <io.h>

int main() {
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* Create a 10-byte file using _dos_creat. */
  if (_dos_creat("DUMMY.FIL", _A_NORMAL, &handle)
      != 0) {
    perror("Unable to _dos_creat DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle, buf, strlen(buf), &count)
      != 0) {
```

```
    perror("Unable to _dos_write to DUMMY.FIL");
    return 1;
  }
  _dos_close(handle);

  /* Create another 10-byte file using _creat. */
  if ((handle = _creat("DUMMY2.FIL", 0)) < 0) {
    perror("Unable to _create DUMMY2.FIL");
    return 1;
  }
  if (_write(handle, buf, strlen(buf)) < 0) {
    perror("Unable to _write to DUMMY2.FIL");
    return 1;
  }
  _close(handle);
  return 0;
}
```

creat

Funktion

Erzeugt eine neue Datei oder überschreibt eine bereits vorhandene Datei.

Syntax

```
#include <sys\stat.h>
int creat(const char *path, int amode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

creat prüft, ob die durch *path* bezeichnete Datei bereits existiert – wenn ja, wird sie auf Null Bytes Länge verkürzt (d.h. der vorige Inhalt gelöscht), für diese Datei definierten Attribute bleiben unverändert (*amode* wird nicht benutzt).

Mit *creat* erzeugte Dateien werden grundsätzlich in dem durch die globale Variable *_fmode* gesetzten Modus (O_TEXT oder O_BINARY) eröffnet.

Existiert eine Datei des angegebenen Namens noch nicht, wird sie von *creat* erzeugt. Wenn die Datei bereits existiert und das Schreiben-Attribut gesetzt ist, wird sie auf 0 Bytes verkürzt und die Dateiattribute bleiben unverändert. Falls bei der Datei das Nur-Lesen-Attribut gesetzt ist, bricht *creat* mit einem Fehler ab, die Datei bleibt unverändert.

Ein Aufruf von *creat* überprüft nur das S_IWRITE-Bit von *amode*. Ist dieses Bit 1, kann in die Datei geschrieben werden, ist das Bit 0, ist nur das Lesen erlaubt. Alle anderen DOS-Dateiattribute werden auf 0 gesetzt.

Die folgenden Konstanten sind in sys\stat.h für *amode* definiert:

Wert von *amode*	Zugriffsberechtigung
S_IWRITE	Schreiben erlaubt
S_IREAD	Lesen erlaubt
S_IREAD| S_IWRITE	Lesen und Schreiben erlaubt

Hinweis: Unter DOS impliziert die Erlaubnis zum Schreiben eine Leseerlaubnis.

Rückgabewert Wenn die angegebene Datei fehlerfrei geöffnet werden konnte, liefert *creat* das neue Datei-Handle (einen positiven Integerwert) zurück.

Im Fall eines Fehlers wird -1 geliefert und *errno* bekommt einen der folgenden Werte:

ENOENT	Suchweg/Datei nicht gefunden
EMFILE	Zu viele Dateien bereits offen
EACCES	Zugriff nicht erlaubt

Siehe auch *chmod*, *chsize*, *close*, *_creat*, *creatnew*, *creattemp*, *dup*, *dup2*, *_fmode* (globale Variable), *fopen*, *open*, *sopen*, *write*

Beispiel

```
#include <sys\stat.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* change default file mode from text to binary */
   _fmode = O_BINARY;

   /* create a binary file for reading and writing */
   handle = creat("DUMMY.FIL", S_IREAD |S_IWRITE);

   /* write 10 bytes to the file */
   write(handle, buf, strlen(buf));

   /* close the file */
   close(handle);
   return 0;
}
```

creatnew

Funktion Erzeugt eine neue Datei.

Syntax

```
#include <dos.h>
int creatnew(const char *path, int mode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

creatnew arbeitet wie *_creat* – mit einer Ausnahme: wenn die angegebene Datei bereits existiert, wird mit einer Fehlermeldung abgebrochen, die Datei bleibt unverändert.

Der Parameter *mode* kann eine der folgenden Konstanten sein, die in dos.h definiert sind:

FA_RDONLY	Nur-Lesen-Attribut
FA_HIDDEN	Verborgene Datei
FA_SYSTEM	Systemdatei

Rückgabewert

Wenn die angegebene Datei fehlerfrei geöffnet werden konnte, liefert *creatnew* das neue Datei-Handle (einen positiven Integerwert) zurück. Im Fall eines Fehlers wird -1 zurückgeliefert und *errno* bekommt einen der folgenden Werte:

EEXIST	Datei existiert bereits
ENOENT	Suchweg/Datei nicht gefunden
EMFILE	Zu viele Dateien bereits offen
EACCES	Zugriff nicht erlaubt

Siehe auch

close, *_creat*, *creat*, *creattemp*, *dup*, *_fmode* (globale Variable), *open*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <dos.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* attempt to create a file */
   /* that doesn't already exist */
   handle = creatnew("DUMMY.FIL", 0);

   if (handle == -1)
      printf("DUMMY.FIL already exists.\n");
   else
   {
      printf("DUMMY.FIL successfully created.\n");
      write(handle, buf, strlen(buf));
      close(handle);
   }
   return 0;
}
```

creattemp

Funktion
Erzeugt eine neue Datei im angegebenen Verzeichnis, wobei die Namensvergabe automatisch erfolgt.

Syntax
```
#include <dos.h>
int creattemp(char *path, int attrib);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung
Mit *creattemp* erzeugte Dateien werden grundsätzlich in dem durch die globale Variable *_fmode* gesetzten Modus (O_TEXT oder O_BINARY) eröffnet.

Das als *path* übergebene Argument bezeichnet ein Verzeichnis, endet mit einem umgekehrten Schrägstrich und sollte lang genug für den resultierenden Dateinamen sein. Durch *creattemp* erzeugte Dateien werden nach dem Ablauf des Programms *nicht* automatisch gelöscht.

creattemp durchsucht das angegebene Verzeichnis und erzeugt eine Datei, deren Name innerhalb dieses Verzeichnisses eindeutig ist, **path* wird danach auf den Namen dieser Datei gesetzt.

Der Funktion *creattemp* kann mit *attrib* ein DOS-Dateiattribut-WORD übergeben werden. Dateien werden grundsätzlich im binären Modus eröffnet. Nach einer fehlerfreien Erzeugung zeigt der Dateizeiger auf das erste Byte der Datei. Die Datei wird für Lese- und Schreibzugriffe geöffnet.

Der Parameter *attrib* kann sich aus den folgenden Werten zusammensetzen, die in dos.h definiert sind:

FA_RDONLY	Nur-Lesen-Attribut
FA_HIDDEN	Verborgene Datei
FA_SYSTEM	Systemdatei

Rückgabewert
Wenn die angegebene Datei fehlerfrei erstellt werden konnte, liefert *creattemp* das neue Datei-Handle (einen positiven Integerwert) zurück.

Im Fall eines Fehlers wird -1 zurückgeliefert und *errno* bekommt einen der folgenden Werte:

ENOENT	Suchweg/Datei nicht gefunden
EMFILE	Zu viele Dateien bereits offen
EACCES	Zugriff nicht erlaubt

Siehe auch
close, *_creat*, *creat*, *creatnew*, *dup*, *_fmode* (globale Variable), *open*

Beispiel
```
#include <string.h>
#include <stdio.h>
#include <io.h>
```

```
int main(void)
{
   int handle;
   char pathname[128];

   strcpy(pathname, "\\");

   /* create a unique file in the root directory */
   handle = creattemp(pathname, 0);

   printf("%s was the unique file created.\n",
          pathname);
   close(handle);
   return 0;
}
```

cscanf

Funktion

Liest und formatiert Eingaben von der Tastatur.

Syntax

```
#include <conio.h>
int cscanf(char *format[,address, ...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

cscanf liest Eingaben in formatierter Form und erwartet zumindest einen Formatstring (Parameter *format*), über den die Anzahl der Eingaben und ihr Format festgelegt wird.

Gelesene Eingaben werden in den Variablen der Parameterliste gespeichert: Nach der Analyse der ersten Formatanweisung wird eine Eingabe gelesen, konvertiert und im korrespondierenden Parameter gespeichert; danach wird der Formatstring auf die nächste Anweisung hin analysiert usw. Die gelesenen Zeichen werden sofort am Bildschirm als »Echo« ausgegeben. Die Anzahl der übergebenen Argumente und der Formatbezeichner muß übereinstimmen.

Hinweis: Nähere Erläuterungen zu Formatbezeichnern finden Sie in der Beschreibung zu *scanf*.

cscanf kann das Lesen eines bestimmten Feldes – d.h. bevor das Ende des Feldes, normalerweise ein »Whitespace«, erreicht ist – oder den Lesevorgang komplett abbrechen. Die möglichen Gründe hierfür werden unter *scanf* erläutert.

Rückgabewert

cscanf liefert die Anzahl der fehlerfrei gelesenen Felder zurück, wobei mit * unterdrückte Felder nicht gezählt werden. Wenn überhaupt keine Zuweisungen erfolgt sind, ist der Rückgabewert 0.

Der Versuch, über das Ende einer Datei hinauszulesen, liefert den Wert EOF.

Siehe auch *fscanf*, *getche*, *scanf*, *sscanf*

Beispiel

```
#include <conio.h>

int main(void)
{
   char string[80];

   /* clear the screen */
   clrscr();

   /* Prompt the user for input */
   cprintf("Enter a string with no spaces:");

   /* read the input */
   cscanf("%s", string);

   /* display what was read */
   cprintf("\r\nThe string entered is: %s", string);
   return 0;
}
```

ctime

Funktion Konvertiert Datum und Uhrzeit in einen String.

Syntax

```
#include <time.h>
char *ctime(const time_t *time);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *ctime* erwartet einen Zeiger auf Datum und Uhrzeit des Systems (das ist der Wert, der von der Funktion *time* geliefert wird). Das Ergebnis von *ctime* ist ein String mit 26 Zeichen, der mit einem Zeilenvorschub und einem Nullzeichen abgeschlossen ist:

```
Mon Nov 21 11:31:54 1983\n\0
```

Alle Felder dieses Strings haben eine konstante Länge.

Speichern Sie in der globalen **long**-Variablen *timezone* den Unterschied zwischen der lokalen Zeit und der *Greenwich Mean Time* (GMT) in Sekunden (für MEZ ist *timezone* -1*60*60). Die globale Variable *daylight* darf nur dann auf einen Wert ungleich Null gesetzt werden, wenn die Standardkonvertierung der USA für Sommer- und Winterzeit angewendet werden soll.

Rückgabewert *ctime* liefert einen Zeiger auf einen statischen String zurück, der mit jedem Aufruf dieser Funktion erneut gesetzt wird.

Siehe auch *asctime*, *daylight* (globale Variable), *difftime*, *ftime*, *getdate*, *gmtime*, *localtime*, *settime*, *time*, *timezone* (globale Variable), *tzset*

Beispiel

```
#include <stdio.h>
#include <time.h>

int main(void)
{
   time_t t;

   time(&t);
   printf("Today's date and time: %s\n", ctime(&t));
   return 0;
}
```

ctrlbrk

Funktion Setzt die Behandlungsroutine für Strg Untbr .

Syntax

```
#include <dos.h>
void ctrlbrk(int (*handler)(void));
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *ctrlbrk* definiert die Funktion, auf die *handler* zeigt, als neue Behandlungsroutine von Strg Untbr . Der Interrupt-Vektor 0x23 wird so modifiziert, daß er die angegebene Funktion aufruft. *ctrlbrk* etabliert einen neuen DOS-Interrupt-Handler, der die angegebene Funktion aufruft; diese Funktion wird nur indirekt aufgerufen.

Über *ctrlbrk* gesetzte C-Routinen können beliebige Schritte und Systemaufrufe enthalten, anstelle eines Rücksprungs ist auch die Verzweigung zu anderen Programmteilen (mit *longjmp*) möglich.

Die durch Strg Untbr aufgerufene Routine kann entweder 0 (Abbruch des Programms) oder einen Wert ungleich 0 (Fortsetzung des Programms) zurückliefern.

Rückgabewert Keiner.

Siehe auch *getcbrk*, *signal*

Beispiel

```
#include <stdio.h>
#include <dos.h>

#define ABORT 0

int c_break(void)
```

```
{
   printf("Control-Break pressed.  "
          "Program aborting ...\n");
   return (ABORT);
}

void main(void)
{
   ctrlbrk(c_break);
   for(;;)
   {
      printf("Looping... Press <Ctrl-Break> "
             "to quit:\n");
   }
}
```

delay

Funktion — Verzögert die Ausführung um die angegebene Anzahl von Millisekunden.

Syntax

```
#include <dos.h>
void delay(unsigned milliseconds);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Mit einem Aufruf von *delay* wird die Ausführung des laufenden Programms für die mit dem Argument *milliseconds* angegebene Zahl von Millisekunden unterbrochen. Die Durchführung einer Kalibrierung vor dem Aufruf von *delay* ist nicht mehr erforderlich. *delay* arbeitet auf die Millisekunde genau.

Rückgabewert — Keiner.

Siehe auch — *nosound, sleep, sound*

Beispiel

```
/* Emits a 440-Hz tone for 500 milliseconds */
#include <dos.h>

int main(void)
{
   sound(440);
   delay(500);
   nosound();

   return 0;
}
```

delline

Funktion Entfernt eine Zeile im Textfenster.

Syntax
```
#include <conio.h>
void delline(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *delline* löscht die Zeile, in der sich der Cursor befindet, und rollt danach alle folgenden Zeilen um eine Position nach oben. *delline* arbeitet im aktuell aktiven Textfenster.

Rückgabewert Keiner.

Siehe auch *clreol*, *clrscr*, *insline*, *window*

Beispiel
```
#include <conio.h>
int main(void)
{
   clrscr();
   cprintf("The function DELLINE deletes the line "
           "containing the\r\n");
   cprintf("cursor and moves all lines below it one "
           "line up.\r\n");
   cprintf("DELLINE operates within the currently "
           "active text\r\n");
   cprintf("window.  Press any key to continue ...");
   gotoxy(1,2);  /* Move the cursor to the second */
                 /* line and first column */
   getch();

   delline();
   getch();

   return 0;
}
```

detectgraph

Funktion Prüft die Hardware des Computers und bestimmt, welcher Grafiktreiber und welcher Grafikmodus benutzt wird.

Syntax
```
#include <graphics.h>
void far detectgraph(int far *graphdriver,
                     int far *graphmode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

detectgraph prüft den Video-Adapter des Computers und setzt die als *graphdriver* und *graphmode* übergebenen Variablen auf die Kennziffer des passenden Grafiktreibers bzw. den Modus mit der höchstmöglichen Auflösung für diesen Treiber.

Wenn kein grafikfähiger Adapter gefunden werden konnte, bekommt **graphdriver* den Wert -2 (grNotDetected) und *graphresult* liefert -2 (grNotDetected) zurück.

**graphdriver* ist die Kennziffer des Grafiktreibers (ein Integerwert). In graphics.h sind folgenden Konstanten für **graphdriver* definiert:

Name	Wert	
DETECT	0	(automatische Erkennung)
CGA	1	
MCGA	2	
EGA	3	
EGA64	4	
EGAMONO	5	
IBM8514	6	
HERCMONO	7	
ATT400	8	
VGA	9	
PC3270	10	

**graphmode* ist ein Integer, der den anfänglichen Grafikmodus festlegt. Wenn **graphdriver* auf DETECT gesetzt ist, wird **graphmode* automatisch auf die höchstmögliche Auflösung für den erkannten Grafiktreiber gesetzt.

Für **graphmode* – den zu setzenden Grafikmodus – sind die folgenden Konstanten in graphics.h definiert:

Grafikmodus	Wert	Spalten * Zeilen	Palette	Seiten
Grafiktreiber CGA				
CGAC0	0	320*200	C0	1
CGAC1	1	320*200	C1	1
CGAC2	2	320*200	C2	1
CGAC3	3	320*200	C3	1
CGAHI	4	640*200	2 Farben	1
Grafiktreiber MCGA				
MCGAC0	0	320*200	C0	1
MCGAC1	1	320*200	C1	1
MCGAC2	2	320*200	C2	1
MCGAC3	3	320*200	C3	1
MCGAMED	4	640*200	2 Farben	1
MCGAHI	5	640*480	2 Farben	1

Grafikmodus	Wert	Spalten * Zeilen	Palette	Seiten
Grafiktreiber EGA				
EGALO	0	640*200	16 Farben	4
EGAHI	1	640*350	16 Farben	2
Grafiktreiber EGA64				
EGA64LO	0	640*200	16 Farben	1
EGA64HI	1	640*350	4 Farben	1
Grafiktreiber EGAMONO				
EGAMONOHI	3	640*350	2 Farben	1*
EGAMONOHI	3	640*350	2 Farben	2**
Grafiktreiber HERC				
HERCMONOHI	0	720*348	2 Farben	2
Grafiktreiber ATT400				
ATT400C0	0	320*200	C0	1
ATT400C1	1	320*200	C1	1
ATT400C2	2	320*200	C2	1
ATT400C3	3	320*200	C3	1
ATT400MED	4	640*200	2 Farben	1
ATT400HI	5	640*400	2 Farben	1
Grafiktreiber VGA				
VGALO	0	640*200	16 Farben	2
VGAMED	1	640*350	16 Farben	2
VGAHI	2	640*480	16 Farben	1
Grafiktreiber PC3270				
PC3270HI	0	720*350	2 Farben	1
Grafiktreiber IBM8514				
IBM8514LO	0	640*480	256 Farben	
IBM8514HI	1	1024*768	256 Farben	

* 64 KByte auf monochromer EGA-Karte
** 256 KByte auf monochromer EGA-Karte

Hinweis: detectgraph wird direkt aufgerufen, um den *initgraph* empfohlenen Grafikmodus zu überschreiben.

Rückgabewert Keiner.

Siehe auch *graphresult*, *initgraph*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* names of the various cards supported */
```

```
char *dname[] = { "requests detection",
                  "a CGA",
                  "an MCGA",
                  "an EGA",
                  "a 64K EGA",
                  "a monochrome EGA",
                  "an IBM 8514",
                  "a Hercules monochrome",
                  "an AT&T 6300 PC",
                  "a VGA",
                  "an IBM 3270 PC"
                };

int main(void)
{
   /* returns detected hardware info. */
   int gdriver, gmode, errorcode;

  /* detect graphics hardware available */
   detectgraph(&gdriver, &gmode);

   /* read result of detectgraph call */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error
                              occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* display the information detected */
   clrscr();
   printf("You have %s video display card.\n",
          dname[gdriver]);
   printf("Press any key to halt:");
   getch();
   return 0;
}
```

difftime

Funktion Berechnet die Differenz zwischen zwei Zeitangaben.

Syntax
```
#include <time.h>
double difftime(time_t time2, time_t time1);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *difftime* berechnet die zwischen *time1* und *time2* liegende Zeit in Sekunden.

Rückgabewert *difftime* liefert die Differenz zweier Zeiten als **double**-Wert zurück.

Siehe auch *asctime*, *ctime*, *daylight* (globale Variable), *gmtime*, *localtime*, *time*, *timezone* (globale Variable)

Beispiel

```
#include <time.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>

int main(void)
{
   time_t first, second;

   clrscr();
   first = time(NULL);  /* Gets system
                           time */
   delay(2000);         /* Waits 2 secs */
   second = time(NULL); /* Gets system time
                           again */

   printf("The difference is: %f seconds\n",
          difftime(second,first));
   getch();

   return 0;
}
```

disable, _disable, enable, _enable

Funktion Deaktiviert und aktiviert Hardware-Interrupts.

Syntax

```
#include <dos.h>
void disable(void);
void _disable(void);
void enable(void);
void _enable(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Diese Makros stellen dem Programmierer eine flexible Steuerung der Hardware-Interrupts zur Verfügung.

Die Makros *disable* und *_disable* deaktivieren alle Hardware-Interrupts (außer NMI: non-maskable interrupt, für Peripheriegeräte).

enable und *_enable* aktivieren alle Hardware-Interrupts.

Rückgabewert Keiner.

Siehe auch *getvect*

Beispiel

```
/* NOTE: This is an interrupt service
 routine. You cannot compile this program
 with Test Stack Overflow turned on and
 get an executable file that operates
 correctly. */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* The clock tick
                        interrupt */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*oldhandler)(__CPPARGS);

int count=0;

void interrupt handler(__CPPARGS)
     /* if C++, need the the ellipsis */
{
/* disable interrupts during the handling */
/* of the interrupt */
   disable();
/* increase the global counter */
   count++;
/* reenable interrupts at the end of the handler */
   enable();
/* call the old routine */
   oldhandler();
}

int main(void)
{
/* save the old interrupt vector */
   oldhandler = getvect(INTR);
```

```
/* install the new interrupt handler */
   setvect(INTR, handler);

/* loop until the counter exceeds 20 */
   while (count < 20)
      printf("count is %d\n",count);

/* reset the old interrupt handler */
   setvect(INTR, oldhandler);

   return 0;
}
```

div

Funktion

Dividiert zwei ganzzahlige Werte und liefert den Quotienten und den Divisionsrest zurück.

Syntax

```
#include <stdlib.h>
div_t div(int numer, int denom);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung

div dividiert die als *numer* (Dividend) und *denom* (Divisor) übergebenen Integerwerte und liefert Quotienten und Divisionsrest in einer Struktur des Typs *div_t* zurück. Dieser Datentyp ist in stdlib.h folgendermaßen definiert:

```
typedef struct {
   int quot;      /* Quotient (Divisionsergebnis) */
   int rem;       /* Divisionsrest */
} div_t;
```

Rückgabewert

div liefert eine Struktur mit den Elementen *quot* (Quotient) und *rem* (Divisionsrest) zurück.

Siehe auch

ldiv

Beispiel

```
/* div example */

#include <stdlib.h>
#include <stdio.h>

div_t x;

int main(void)
{
   x = div(10,3);
   printf("10 div 3 = %d remainder %d\n",
   x.quot, x.rem);
   return 0;
}
```

Programm-ausgabe:
```
10 div 3 = 3 remainder 1
```

_dos_close

Funktion: Schließt eine Datei.

Syntax:
```
#include <dos.h>
unsigned dos_close(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung: *_dos_close* schließt die durch *handle* bezeichnete Datei. *handle* ist ein Datei-Handle, das durch einen Aufruf von *_dos_creat*, *_dos_creatnew* oder *_dos_open* einer Datei zugeordnet wurde.

Rückgabewert: *_dos_close* liefert den Wert 0 zurück, wenn die Operation fehlerfrei ausgeführt werden konnte. Andernfalls liefert die Funktion den DOS-Fehlercode, und die globale Variable *errno* bekommt folgenden Wert:

EBADF	Ungültige Dateinummer

Siehe auch: *_dos_creat*, *_dos_open*, *_dos_read*, *_dos_write*

Beispiel:
```
#include <dos.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  if (_dos_creat("DUMMY.FIL",_A_NORMAL, &handle) != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle,buf,strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }
  /* close the file */
  _dos_close(handle);
  return 0;
}
```

_dos_creatnew

Funktion Erzeugt eine neue Datei.

Syntax
```
#include <dos.h>
unsigned _dos_creatnew(const char *path, int attrib,
                       int *handlep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_dos_creatnew* erstellt und öffnet mit Hilfe der DOS-Funktion 0x5B die durch *path* bezeichnete neue Datei im binären Modus. Die Datei erhält die Zugriffsberechtigung *attrib* (ein DOS-Attribut-WORD). Nach einer fehlerfreien Erzeugung wird das Datei-Handle in dem durch *handlep* bezeichneten Bereich gespeichert und der Dateizeiger zeigt auf das erste Byte der Datei. Die Datei wird für Lese- und Schreibzugriffe geöffnet.

Wenn die Datei bereits existiert, wird mit einer Fehlermeldung abgebrochen, die Datei bleibt unverändert.

Für *_dos_creatnew* ist der Parameter *attrib* eine ODER-Kombination aus einer oder mehreren der folgenden in dos.h definierten Konstanten:

_A_NORMAL	Normale Datei
_A_RDONLY	Nur-Lesen-Datei
_A_HIDDEN	Verborgene Datei
_A_SYSTEM	Systemdatei

Rückgabewert Wenn die angegebene Datei fehlerfrei geöffnet werden konnte, liefert *_dos_creatnew* 0 zurück. Anderfalls wird der DOS-Fehlercode zurückgeliefert und die globale Variable *errno* bekommt einen der folgenden Werte:

EEXIST	Datei existiert bereits
ENOENT	Suchweg/Datei nicht gefunden
EMFILE	Zu viele Dateien bereits offen
EACCES	Zugriff nicht erlaubt

Siehe auch *_dos_close*, *_dos_creat*, *_dos_getfileattr*, *_dos_setfileattr*

Beispiel
```
#include <dos.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
  unsigned count;
  int handle;
  char buf[11] = "0123456789";
```

```
  /* create a file containing 10 bytes */
  if (_dos_creatnew("DUMMY.FIL", _A_NORMAL, &handle)
      != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle,buf,strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }
  /* close the file */
  _dos_close(handle);
  return 0;
}
```

dosexterr

Funktion — Liefert einen erweiterten Fehlercode von DOS zurück.

Syntax

```
#include <dos.h>
int dosexterr(struct DOSERROR *eblkp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung — *dosexterr* verwendet die DOS-Funktion 0x59, um in der Struktur DOSERROR nach einer fehlerhaften DOS-Operation zusätzliche Informationen zu liefern. Der Strukturtyp DOSERROR ist in dos.h folgendermaßen definiert:

```
struct DOSERROR {
   int de_exterror;    /* Fehlernummer */
   char de_class;      /* Fehlerklasse */
   char de_action;     /* Aktion */
   char de_locus;      /* Fehlerstelle */
};
```

Ein Wert von 0 für *de_exterror* zeigt an, daß die zuletzt ausgeführte DOS-Operation fehlerfrei verlaufen ist.

Rückgabewert — *dosexterr* liefert den Wert von *de_exterror* zurück.

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   FILE *fp;
```

```
    struct DOSERROR info;

    fp = fopen("perror.dat","r");
    if (!fp) perror("Unable to open file for reading");
    dosexterr(&info);

    printf("Extended DOS error information:\n");
    printf("   Extended error: %d\n",info.de_exterror);
    printf("            Class: %x\n",info.de_class);
    printf("           Action: %x\n",info.de_action);
    printf("      Error Locus: %x\n",info.de_locus);

    return 0;
}
```

_dos_findfirst

Funktion

Sucht ein Dateiverzeichnis ab.

Definition

```
#include <dos.h>
unsigned _dos_findfirst(attrib, const char *pathname,
                        int struct find_t *ffblk);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_dos_findfirst verwendet die DOS-Funktion *0x4E*, um ein Verzeichnis nach dem ersten Vorkommen eines Dateieintrags abzusuchen.

Der Parameter *pathname* ist ein String und bezeichnet den zu suchenden Eintrag. Er kann optional eine Laufwerksbezeichnung und/oder einen Suchweg enthalten, Jokerzeichen (* und ?) sind erlaubt. Falls eine Datei gefunden wird, werden in der über *ffblk* angegebenen Struktur *find_t* die Datei- und Verzeichnisinformationen gespeichert.

Die Struktur *find_t* ist folgendermaßen definiert:

```
struct find_t {
    char reserved[21];      /* von DOS verwendet */
    char attrib;            /* Attribut */
    int wr_time;            /* Erstellungszeit */
    int wr_date;            /* Datum der Erstellung */
    long size;              /* Dateigröße in Bytes */
    char name[13];          /* Name der Datei */
};
```

Über den Parameter *attrib* wird das Attribut des zu suchenden Eintrags angegeben. Der Parameter *attrib* enthält eine

ODER-Kombination aus einer oder mehreren der folgenden in dos.h definierten Konstanten:

_A_NORMAL	Normale Datei
_A_RDONLY	Nur-Lesen-Attribut
_A_HIDDEN	Verborgene Datei
_A_SYSTEM	Systemdatei
_A_VOLID	Name des Datenträgers
_A_SUBDIR	Verzeichnis
_A_ARCH	Archiv-Bit

Ausführliche Informationen über diese Attribute finden Sie in Ihren DOS-Handbüchern.

Beachten Sie, daß *wr_time* und *wr_date* Bitfelder für Erstellungszeit und Erstellungsdatum der Datei enthalten. Die Struktur dieser Bitfelder ist durch DOS vorgegeben.

Es handelt sich in beiden Fällen um 16 Bit große Strukturen, die in drei Felder unterteilt sind:

wr_time:

Bits 0 bis 4	Resultierende Sekunden geteilt durch 2 (d.h. 10 bedeutet hier 20 Sekunden)
Bits 5 bis 10	Minuten
Bits 11 bis 15	Stunden

wr_date:

Bits 0 bis 4	Tag
Bits 5 bis 8	Monat
Bits 9 bzw. 15	Jahre seit 1980 (d.h. 9 bedeutet hier 1989)

Rückgabewert *_dos_findfirst* liefert den Wert 0 zurück, wenn ein zu *pathname* passender Eintrag gefunden wurde.

Wenn kein Eintrag gefunden wird oder der angegebene Dateiname aus irgendeinem Grund ungültig ist, wird der DOS-Fehlercode als Ergebnis geliefert und *errno* bekommt den Wert:

ENOENT	Datei/Suchweg nicht gefunden

Siehe auch *_dos_findnext*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  struct find_t ffblk;
  int done;
  printf("Directory listing of *.*\n");
  done = _dos_findfirst("*.*",_A_NORMAL,&ffblk);
  while (!done) {
    printf("  %s\n", ffblk.name);
    done = _dos_findnext(&ffblk);
```

```
  }
  return 0;
}
```

Programm-ausgabe:

```
Directory listing of *.*
   FINDFRST.C
   FINDFRST.OBJ
   FINDFRST.MAP
   FINDFRST.EXE
```

_dos_findnext

Funktion: Setzt eine *_dos_findfirst*-Suche fort.

Definition:

```
#include <dos.h>
unsigned _dos_findnext(struct find_t *ffblk);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung: Mit *_dos_findnext* kann ein Verzeichnis nach weiteren Einträgen abgesucht werden, die zu *pathname* einer vorangegangenen *_dos_findfirst*-Suche passen. *ffblk* zeigt auf die gleiche Struktur wie bei *_dos_findfirst* und enthält die notwendigen Informationen zur Fortsetzung der Suche. *_dos_findnext* liefert bei jedem Aufruf einen weiteren Dateieintrag zurück – solange, bis das Verzeichnis keine weiteren Einträge mehr enthält, die *pathname* von *_dos_findfirst* entsprechen.

Rückgabewert: *_dos_findnext* liefert den Wert 0 zurück, wenn ein weiterer, zu *pathname* passender Eintrag gefunden wurde. Sind keine weiteren Einträge vorhanden oder ist der angegebene Dateiname aus irgendeinem Grund ungültig, wird der DOS-Fehlercode zurückgeliefert und *errno* bekommt den Wert:

ENOENT Datei/Suchweg nicht gefunden

Siehe auch: *_dos_findfirst*

Beispiel:

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  struct find_t ffblk;
  int done;
  printf("Directory listing of *.*\n");
  done = _dos_findfirst("*.*",_A_NORMAL,&ffblk);
  while (!done) {
    printf("  %  s\n", ffblk.name);
```

```
    done = _dos_findnext(&ffblk);
  }
  return 0;
}
```

Programm-ausgabe:

```
Directory listing of *.*
  FINDFRST.C
  FINDFRST.OBJ
  FINDFRST.MAP
  FINDFRST.EXE
```

_dos_getdiskfree

Funktion: Ermittelt den freien Platz auf einer Diskette/Festplatte.

Syntax:

```
#include <dos.h>
unsigned _dos_getdiskfree(unsigned char drive,
                          struct diskfree_t *dtable);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung: *_dos_getdiskfree* ermittelt Informationen über den freien Platz auf dem durch *drive* angegebenen Laufwerk und speichert sie in der durch *dtable* bezeichneten Struktur des Typs *diskfree_t*.

Wenn *drive* den Wert 0 hat, wird das aktuelle Laufwerk bearbeitet, ansonsten gilt A = 1, B = 2 usw.

Der Strukturtyp *diskfree_t* ist folgendermaßen definiert:

```
struct diskfree_t {
   unsigned avail_clusters;      /* freie Cluster */
   unsigned total_clusters;      /* Gesamtzahl Cluster
*/
   unsigned bytes_per_sector;    /* Bytes pro Sektor */
   unsigned sectors_per_cluster; /* Sektoren pro  */
                                 /* Cluster */
};
```

Rückgabewert: *_dos_getdiskfree* liefert bei fehlerfreier Ausführung 0 zurück. Andernfalls wird ein Wert ungleich Null zurückgeliefert und *errno* bekommt den Wert:

EINVAL Ungültiges Laufwerk

Siehe auch: *getfat, getfatd*

Beispiel:

```
#include <stdio.h>
#include <dos.h>
#include <process.h>
```

```
int main(void)
{
  struct diskfree_t free;
  long avail;

  if (_dos_getdiskfree(0, &free) != 0) {
    printf("Error in _dos_getdiskfree() call\n");
    exit(1);
  }
  avail = (long) free.avail_clusters
        * (long) free.bytes_per_sector
        * (long) free.sectors_per_cluster;
  printf("The current drive has %ld bytes "
         "available\n", avail);
  return 0;
}
```

_dos_getdrive, _dos_setdrive

Funktion

Liest bzw. ändert die aktuelle Laufwerksnummer.

Syntax

```
#include <dos.h>
void _dos_getdrive(unsigned *drivep);
void _dos_setdrive(unsigned drivep, unsigned *ndrives);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

_dos_getdrive verwendet die DOS-Funktion 0x19 zur Ermittlung der aktuellen Laufwerksnummer.

_dos_setdrive setzt die aktuelle Laufwerksnummer mit der DOS-Funktion 0x0E und speichert die Gesamtzahl der Laufwerke im Bereich, auf den *ndrives* zeigt.

Für die Laufwerksnummern in dem durch *drivep* bezeichneten Bereich gilt: 1 = A, 2 = B, 3 = C, usw.)

Rückgabewert

Keiner. Mit *_dos_getdrive* können feststellen, ob das aktuelle Laufwerk erfolgreich geändert wurde.

Siehe auch

getcwd

Beispiel

```
#include <stdio.h>
#include <dos.h>
int main(void)
{
  unsigned disk;
  _dos_getdrive(&disk);
  printf("The current drive is: %c\n",disk + 'A' - 1);
  return 0;
}
```

_dos_getfileattr, _dos_setfileattr

Funktion

Ändert den Zugriffsmodus einer Datei.

Syntax

```
#include <dos.h>
int _dos_getfileattr(const char *path,
                     unsigned *attribp);
int _dos_setfileattr(const char path, unsigned attrib);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_Dos_getfileattr liest die DOS-Dateiattribute der durch *path* angegebenen Datei. Die Attribute werden in dem durch *attribp* bezeichneten Bereich gespeichert.

_dos_setfileattr setzt die Zugriffsberechtigung der durch *path* angegebenen Datei auf den Wert von *attrib*. Die DOS-Dateiattribute bestehen aus einer ODER-Kombination der folgenden in dos.h definierten symbolischen Konstanten:

_A_RDONLY	Nur-Lesen-Attribut
_A_HIDDEN	Verborgene Datei
_A_SYSTEM	Systemdatei
_A_VOLID	Datenträgername
_A_SUBDIR	Verzeichnis
_A_ARCH	Archiv-Bit
_A_NORMAL	Normale Datei (keine Attribut-Bits gesetzt)

Rückgabewert

Bei fehlerfreier Ausführung liefern *_dos_getfileattr* und *_dos_getfileattr* als Ergebnis 0. Andernfalls wird der DOS-Fehlercode zurückgeliefert und die globale Variable *errno* bekommt den Wert:

ENOENT	Suchweg/Datei nicht gefunden

Siehe auch

chmod, stat

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  char filename[128];
  unsigned attrib;
  printf("Enter a file name:");
  scanf("%s", filename);
  if (_dos_getfileattr(filename,&attrib) != 0)
  {
    perror("Unable to obtain file attributes");
    return 1;
  }
```

```
  if (attrib & _A_RDONLY)
    printf("%s is read-only.\n", filename);

  if (attrib & _A_HIDDEN)
    printf("%s is hidden.\n", filename);

  if (attrib & _A_SYSTEM)
    printf("%s is a system file.\n", filename);

  if (attrib & _A_VOLID)
    printf("%s is a volume label.\n", filename);

  if (attrib & _A_SUBDIR)
    printf("%s is a directory.\n", filename);

  if (attrib & _A_ARCH)
    printf("%s is an archive file.\n", filename);
  return 0;
}
```

_dos_getftime, _dos_setftime

Funktion

Liest bzw. setzt Datum und Uhrzeit einer Datei.

Syntax

```
#include <dos.h>
unsigned _dos_getftime(int handle, unsigned *datep,
                       unsigned *timep);
unsigned _dos_setftime(int handle, unsigned date,
                       unsigned time);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_dos_getftime liest die Uhrzeit und das Datum der durch *handle* angegebenen Datei. Diese Datei muß vorher mit *_dos_open*, *_dos_creat* oder *_dos_creatnew* geöffnet worden sein. *_dos_getftime* speichert das Datum und die Zeit in den durch *datep* und *timep* bezeichneten Bereichen.

_dos_setftime setzt Datum und Uhrzeit der Datei auf die neuen, in den Parametern *time* und *date* angegebenen Werte.

Beachten Sie, daß die Datums- und Uhrzeitwerte Bitfelder für die Uhrzeit und das Datum der Datei enthalten.

Die Struktur dieser Bitfelder ist durch DOS vorgegeben. Es handelt sich in beiden Fällen um 16 Bit große Strukturen, die in drei Felder unterteilt sind:

Datum:

Bits 0 bis 4	Tag
Bits 5 bis 8	Monat
Bits 9 bis 15	Jahre seit 1980 (d.h. 9 bedeutet hier 1989)

Uhrzeit:

Bits 0 bis 4	Resultierende Sekunden geteilt durch 2 (d.h. 10 bedeutet hier 20 Sekunden)
Bits 5 bis 10	Minuten
Bits 11 bis 15	Stunden

Rückgabewert *_dos_getftime* und *_dos_setftime* liefern bei fehlerfreier Ausführung 0 zurück.

Andernfalls wird der DOS-Fehlercode zurückgeliefert und die globale Variable *errno* bekommt den Wert:

EBADF	Ungültige Dateinummer

Siehe auch *fstat, stat*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main()
{
  FILE *stream;
  unsigned date, time;
  if ((stream = fopen("TEST.$$$", "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    return 1;
  }
  _dos_getftime(fileno(stream), &date, &time);
  printf("File date: 0x%x\n",date);
  printf("File time: 0x%x\n",time);
  fclose(stream);
  return 0;
}
```

_dos_gettime, _dos_settime

Funktion Liest und ändert die Systemzeit.

Syntax

```
#include <dos.h>
void _dos_gettime(struct dostime_t *timep);
unsigned _dos_settime(struct dostime_t *timep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_dos_gettime* speichert die die aktuelle Systemzeit in der über *timep* angegebenen Struktur *dostime_t*.

_dos_settime setzt die Systemzeit auf die Werte in der Struktur *dostime_t*, auf die *timep* zeigt.

Der Strukturtyp *dostime_t* ist folgendermaßen definiert:

```
struct dostime_t {
   unsigned char hour;     /* Stunden (0 bis 23) */
   unsigned char minute;   /* Minuten (0 bis 59) */
   unsigned char second;   /* Sekunden (0 bis 59) */
   unsigned char hsecond;  /* Hundertstel Sekunden */
                           /* (0 bis 99 */
};
```

Rückgabewert *_dos_gettime* hat keinen Rückgabewert.

Bei fehlerfreier Ausführung liefert *_dos_settime* als Ergebnis 0. Andernfalls wird der DOS-Fehlercode zurückgeliefert und die globale Variable *errno* bekommt den Wert:

EINVAL Ungültige Uhrzeit

Siehe auch *_dos_getdate*, *_dos_setdate*, *_dos_settime*, *stime*, *time*

Beispiel

```
#include   <stdio.h>
#include   <dos.h>

int main(void)
{
  struct dostime_t t;
  _dos_gettime(&t);
  printf("The current time is: "
         "%2d:%02d:%02d.%02d\n", t.hour, t.minute,
         t.second, t.hsecond);
  return 0;
}
```

_dos_getvect

Funktion Liest einen Interrupt-Vektor.

Syntax

```
#include <dos.h>
void interrupt(*_dos_getvect(unsigned interruptno))();
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Die Prozessoren der 8086-Familie reservieren eine Reihe von Interrupt-Vektoren, die von 0 bis 255 durchnumeriert sind. Der vier Byte lange Wert jedes Vektors ist die Adresse einer Interrupt-Funktion.

_dos_getvect liefert den Wert des Vektors *interruptno* als **far** Zeiger auf eine Funktion des Typs **interrupt** zurück. Der Wert von *interruptno* muß zwischen 0 und 255 liegen.

Rückgabewert *_dos_getvect* liefert den aktuellen vier Byte langen Wert zurück, der im Interrupt-Vektor *interruptno* gespeichert ist.

Siehe auch *_disable, _enable, _dos_setvect*

Beispiel

```
#include <stdio.h>
#include <dos.h>

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt get_out(__CPPARGS);
         /* interrupt prototype */
void interrupt (*oldfunc)(__CPPARGS);
         /* interrupt function pointer */

int looping = 1;
int main(void)
{
  puts("Press <Shift><PrtSc> to terminate");
  /* save the old interrupt */
  oldfunc  = _dos_getvect(5);

  /* install interrupt handler */
  _dos_setvect(5,get_out);

  /* do nothing */
  while (looping);

  /* restore to original interrupt routine */
  _dos_setvect(5,oldfunc);

  puts("Success");
  return 0;
}

void interrupt get_out(__CPPARGS) {
  looping = 0;
          /* change global var to get out of loop */
}
```

_dos_setvect

Funktion Setzt den Eintrag eines Interrupt-Vektors.

Syntax

```
#include <dos.h>
void _dos_setvect(unsigned interruptno, void interrupt
                  (*isr)());
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Die Prozessoren der 8086-Familie reservieren eine Reihe von Interrupt-Vektoren, die von 0 bis 255 durchnumeriert sind. Der vier Byte lange Wert jedes Vektors ist die Adresse einer Interrupt-Funktion.

_dos_setvect setzt den Interrupt-Vektor Nummer *interruptno* so, daß er auf die als *isr* (interrupt service routine) angegebene C-Funktion zeigt. Diese C-Funktion muß als **interrupt** -Routine deklariert worden sein.

Hinweis: Solange die in dos.h deklarierten Prototypen verwendet werden, ist *_dos_setvect* ohne weitere Modifikationen in jedem Speichermodell anwendbar.

Rückgabewert Keiner.

Siehe auch *_dos_getvect*

Beispiel

```
#include <stdio.h>
#include <dos.h>

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt get_out(__CPPARGS);
                    /* interrupt prototype */
void interrupt (*oldfunc)(__CPPARGS);
                    /* interrupt function pointer */

int looping = 1;

int main(void)
{
  puts("Press <Shift><PrtSc> to terminate");

  /* save the old interrupt */
  oldfunc  = _dos_getvect(5);

  /* install interrupt handler */
  _dos_setvect(5,get_out);

  /* do nothing */
  while (looping);

  /* restore to original interrupt routine */
  _dos_setvect(5,oldfunc);

  puts("Success");
  return 0;
```

```
}

void interrupt get_out(__CPPARGS) {
  looping = 0;     /* change global var
                      to get out of loop */
}
```

_dos_write

Funktion Schreibt Daten in eine Datei.

Syntax
```
#include <dos.h>
unsigned _dos_write(int handle, void far *buf,
                    unsigned len, unsigned *nwritten);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_dos_write* liest mit Hilfe der DOS-Funktion 0x40 *len* Bytes aus dem Puffer, auf den der **far**-Zeiger *buf* zeigt, und schreibt sie in die durch *handle* bezeichnete Datei.

Da die Funktion *_dos_write* nur mit binären Dateien arbeitet, »übersetzt« sie Zeilenvorschubzeichen (LF) *nicht* in die Zeichenfolge CR/LF (Wagenrücklauf/Zeilenvorschub).

_dos_write speichert die Anzahl der geschriebenen Bytes an der durch *nwritten* bezeichneten Adresse. Werden weniger als *len* Bytes geschrieben, sollte dies als Fehler interpretiert werden (normalerweise steht ein solches Ergebnis für »Diskette voll«).

Bei Diskettendateien schreibt *_dos_write* ab der aktuellen Positionen des Dateizeigers, bei Gerätedateien wird direkt zum jeweiligen Gerät geschrieben.

Rückgabewert *_dos_write* liefert bei fehlerfreier Ausführung das Ergebnis 0. Andernfalls wird der DOS-Fehlercode zurückgeliefert und die globale Variable *errno* bekommt den einen der folgenden Werte:

EACCES	Zugriff nicht gestattet
EBADF	Ungültige Dateinummer

Siehe auch *_dos_open*, *_dos_creat*, *_dos_read*

Beispiel
```
#include <dos.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
  unsigned count;
  int handle;
```

```
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  if (_dos_creat("DUMMY.FIL",_A_NORMAL, &handle) != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle,buf,strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }

  /* close the file */
  _dos_close(handle);
  return 0;
}
```

dostounix

Funktion

Konvertiert Datum und Uhrzeit in das UNIX-Format.

Syntax

```
#include <dos.h>
long dostounix(struct date *d, struct time *t);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

dostounix erwartet einen Zeiger auf eine Struktur des Typs *date* (die beispielsweise von *getdate* gesetzt wird) sowie einen Zeiger auf eine Struktur des Typs *time* (wie sie *gettime* liefert) und konvertiert beide Strukturen in das UNIX-Datumsformat. Die Strukturen *date* und *time* müssen Datum und Uhrzeit im DOS-Format enthalten.

Hinweis: Das Datum muß größer oder gleich 01/01/1980 sein, die Uhrzeit größer oder gleich 00:00:00.

Rückgabewert

dostounix liefert Datum und Uhrzeit im UNIX-Format zurück: die Zahl der Sekunden seit dem 1. Januar 1970, 00:00:00 Uhr (GMT – Greenwich Meantime)

Siehe auch

unixtodos

Beispiel

```
#include <time.h>
#include <stddef.h>
#include <dos.h>
#include <stdio.h>
```

```
int main(void)
{
   time_t t;
   struct time d_time;
   struct date d_date;
   struct tm *local;

   getdate(&d_date);
   gettime(&d_time);

   t = dostounix(&d_date, &d_time);
   local = localtime(&t);
   printf("Time and Date: %s\n", asctime(local));

   return 0;
}
```

drawpoly

Funktion — Zeichnet den Umriß eines Polygons.

Syntax

```
#include <graphics.h>
void far drawpoly(int numpoints, int far *polypoints);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *drawpoly* zeichnet den Umriß eines Polygons mit *numpoints* Eckpunkten in der aktuellen Linienart und -farbe. Der als *polypoints* übergebene Zeiger muß auf eine Folge von Integerwerten (*numpoints* * 2) zeigen. Jedes Integerpaar steht für die *x*- und *y*-Koordinaten eines Polygonpunktes.

Hinweis: Zum Zeichnen einer geschlossenen Figur mit *n* Eckpunkten muß *polypoints n+1* Koordinatenpaare enthalten, wobei das letzte Paar dieselben Werte wie das erste Paar hat.

Rückgabewert — Keiner.

Siehe auch — *fillpoly, floodfill, graphresult, setwritemode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;
```

```
   /* our polygon array */
   int poly[10];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)
   /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
   /* terminate with an error code */
      exit(1);
   }

   maxx = getmaxx();
   maxy = getmaxy();

   poly[0] = 20;        /* 1st vertext */
   poly[1] = maxy / 2;

   poly[2] = maxx - 20; /* 2nd */
   poly[3] = 20;

   poly[4] = maxx - 50; /* 3rd */
   poly[5] = maxy - 20;

   poly[6] = maxx / 2;  /* 4th */
   poly[7] = maxy / 2;
/*
   drawpoly doesn't automatically close
   the polygon, so we close it.
*/
   poly[8] = poly[0];
   poly[9] = poly[1];

   /* draw the polygon */
   drawpoly(5, poly);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

dup

Funktion Verdoppelt ein Handle für eine Datei.

Syntax
```
#include <io.h>
int dup(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *dup* erzeugt aus einem Datei-Handle ein zweites Handle, das mit dem Original folgende Gemeinsamkeiten hat:

- die offene Datei bzw. das Gerät;
- den Dateizeiger (ändert man den Dateizeiger des einen Handle, so ändert man auch den des anderen);
- die Zugriffsberechtigungen (Lesen, Schreiben, Lesen und Schreiben).

handle ist ein Datei-Handle, das man durch einen Aufruf von *_creat*, *creat*, *_open*, *open*, *dup* oder *dup2* erhält.

Rückgabewert *dup* liefert bei fehlerfreier Ausführung das neu erzeugte Datei-Handle als positiven Integerwert zurück.

Im Falle eines Fehlers liefert die Routine den Wert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

EMFILE	Zu viele offene Dateien
EBADF	Ungültiges Handle

Siehe auch *_close*, *close*, *_creat*, *creat*, *creatnew*, *creattemp*, *dup2*, *fopen*, *_open*, *open*

Beispiel
```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

void flush(FILE *stream);

int main(void)
{
   FILE *fp;
   char msg[] = "This is a test";

   /* create a file */
   fp = fopen("DUMMY.FIL", "w");

   /* write some data to the file */
   fwrite(msg, strlen(msg), 1, fp);
```

```
   clrscr();
   printf("Press any key to flush DUMMY.FIL:");
   getch();

  /* flush the data to DUMMY.FIL without closing it */
   flush(fp);

   printf("\nFile was flushed, "
          "Press any key to quit:");
   getch();
   return 0;
}

void flush(FILE *stream)
{
   int duphandle;

   /* flush internal buffer */
   fflush(stream);

   /* make a duplicate file handle */
   duphandle = dup(fileno(stream));

   /* close the duplicate handle */
   /* to flush the DOS buffer */
   close(duphandle);
}
```

dup2

Funktion

Bildet das Datei-Handle (*oldhandle*) auf dem existierenden Datei-Handle (*newhandle*) ab.

Syntax

```
#include <io.h>
int dup2(int oldhandle, int newhandle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

dup2 erzeugt aus einem Handle ein zweites, das mit dem Original folgende Gemeinsamkeiten hat:

- die offene Datei bzw. das Gerät;
- den Dateizeiger (ändert man den Dateizeiger des einen Handle, so ändert man auch den des anderen);
- die Zugriffsberechtigungen (Lesen, Schreiben, Lesen und Schreiben).

dup2 erzeugt ein neues Handle mit dem Wert von *newhandle*. Ist die mit *newhandle* verbundene Datei beim Aufruf von *dup2* geöffnet, wird sie geschlossen. *newhandle* und *old-*

handle sind Datei-Handle, die durch Aufrufe von *creat*, *open*, *dup* oder *dup2* erzeugt wurden.

Hinweis: *dup2* ist unter UNIX-System III nicht verfügbar.

Rückgabewert

dup2 liefert bei fehlerfreier Ausführung den Wert 0 zurück. Im Falle eines Fehlers liefert die Routine den Wert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

EMFILE	Zu viele offene Dateien
EBADF	Ungültiges Handle

Siehe auch

_close, *close*, *_creat*, *creat*, *creatnew*, *creattemp*, *dup*, *fopen*, *_open*, *open*

Beispiel

```
#include <sys\stat.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   #define STDOUT 1

   int nul, oldstdout;
   char msg[] = "This is a test";

   /* create a file */
   nul = open("DUMMY.FIL", O_CREAT | O_RDWR,
      S_IREAD | S_IWRITE);

   /* create a duplicate handle for standard output */
   oldstdout = dup(STDOUT);
   /*
      redirect standard output to DUMMY.FIL
      by duplicating the file handle onto
      the file handle for standard output.
   */
   dup2(nul, STDOUT);

   /* close the handle for DUMMY.FIL */
   close(nul);

   /* will be redirected into DUMMY.FIL */
   write(STDOUT, msg, strlen(msg));

   /* restore original standard output handle */
   dup2(oldstdout, STDOUT);

   /* close duplicate handle for STDOUT */
   close(oldstdout);

   return 0;
}
```

ecvt

Funktion — Konvertiert eine Gleitkommazahl in einen String.

Syntax

```
#include <stdlib.h>
char *ecvt(double value, int ndig, int *dec,
           int *sign);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *ecvt* konvertiert *value* in einen nullterminierten String mit *ndig* Ziffern – beginnend mit der höchstwertigen Ziffer (links) – und liefert einen Zeiger auf diesen String zurück. Der String enthält ausschließlich Ziffern: über *dec* wird die Position des Dezimalpunkts relativ zum Beginn des Strings zurückgeliefert (negative Werte bedeuten hier: »Dezimalpunkt links vom Stringanfang«). Wenn *value* negativ ist, dann setzt *ecvt sign* auf einen Wert ungleich Null (für positive Werte von *value* ist *sign* == 0). Die letzte Dezimalstelle des zurückgelieferten Strings ist gerundet.

Rückgabewert — *ecvt* liefert einen Zeiger auf diesen statischen String zurück, dessen Inhalt durch jeden Aufruf von *fcvt* und *ecvt* gesetzt wird.

Siehe auch — *fcvt, gcvt, sprintf*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   char *string;
   double value;
   int dec, sign;
   int ndig = 10;

   clrscr();
   value = 9.876;
   string = ecvt(value, ndig, &dec, &sign);
   printf("string = %s      dec = %d sign = %d\n",
          string, dec, sign);

   value = -123.45;
   ndig= 15;
   string = ecvt(value,ndig,&dec,&sign);
   printf("string = %s dec = %d sign = %d\n",
          string, dec, sign);

   value = 0.6789e5; /* scientific notation */
   ndig = 5;
```

```
   string = ecvt(value,ndig,&dec,&sign);
   printf("string = %s         dec = %d sign = %d\n",
          string, dec, sign);

   return 0;
}
```

ellipse

Funktion — Zeichnet einen elliptischen Kreisbogen.

Syntax

```
#include <graphics.h>
void far ellipse(int x, int y, int stangle,
                 int endangle, int xradius,
                 int yradius);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *ellipse* zeichnet einen elliptischen Kreisbogen um den Mittelpunkt (*x*,*y*) mit den Radien *xradius* (horizontal) und *yradius* (vertikal). *stangle* und *endangle* legen den Anfangs- bzw. Endpunkt des Kreisbogens fest. Hat *x* den Wert 0 und *y* den Wert 360, wird eine vollständige Ellipse gezeichnet. Die Angabe beider Winkel erfolgt in Grad, wobei entgegen dem Uhrzeigersinn gezählt wird (0 Grad = horizontal rechts des Mittelpunktes, 90 Grad = senkrecht über dem Mittelpunkt usw).

Hinweis: Der Parameter *linestyle* von *setlinestyle* für die Linienart wirkt sich auf *arc, circle, ellipse, sector* und *pieslice* nicht aus; der Parameter *thickness* für die Liniendicke wird dagegen verwendet.

Rückgabewert — Keiner.

Siehe auch — *arc, circle, fillellipse, sector*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 0, endangle = 360;
   int xradius = 100, yradius = 50;

   /* initialize graphics, local variables */
   initgraph(&gdriver, &gmode, "");
```

```
   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)
   /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* draw ellipse */
   ellipse(midx, midy, stangle, endangle,
           xradius, yradius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

__emit__

Funktion — Fügt Literale direkt in den Objektcode ein.

Syntax

```
#include <dos.h>
void __emit__(argument, ... );
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung — *__emit__* erlaubt das direkte Einfügen von Literalen während des Compilierens eines Objektcodes. Die Funktion dient zur Erzeugung von Assembler-Anweisungen ohne Verwendung eines Assemblers. Im allgemeinen wird man als Argumente für *__emit__* einzelne Bytes (d.h. Prozessorbefehle) verwenden. Es lassen sich aber auch wesentlich komplexere Konstrukte bilden – inklusive Zugriffen auf C-Variablen.

Hinweis: Sie sollten diese Funktion nur verwenden, wenn Sie mit der Maschinensprache der 8086-Prozessorfamilie vertraut sind. Sie können mit Hilfe dieser Funktion beliebige Bytes in den Befehlscode einer Funktion einfügen. Ein einziges fehlerhaftes Byte kann zu unerwartetem Verhalten Ihres Pro-

gramms und auch ohne weiteres zu einem Systemabsturz führen. Der Compiler unternimmt keinen Versuch, die eingefügten Bytefolgen auf formale Korrektheit oder in irgendeiner anderen Weise zu prüfen. Der Compiler bemerkt nicht, ob Sie Maschinenbefehle einfügen, die den Inhalt von Prozessorregistern oder des Speichers ändern. Anders als bei der Verwendung von Inline-Assembler, wo er beispielsweise die Benutzung der Register SI und DI erkennt, ergreift der Compiler keine Maßnahmen zur Sicherung von Registerinhalten. Der korrekte Einsatz dieser Funktion liegt allein in der Verantwortung des Programmierers.

__emit__ erwartet zumindest ein Argument, es können aber beliebig viele übergeben werden. Die Argumente für diese Funktion werden nicht wie Argumente bei normalen Funktionsaufrufen behandelt, da keinerlei Konvertierungen durchgeführt werden.

Die Parameter dieser Funktion unterliegen denselben Beschränkungen wie Initialisierungswerte statischer Objekte: es dürfen Integer- und Gleitkommakonstanten und die Adressen statischer Objekte verwendet werden. Die Werte dieser Ausdrücke werden exakt an der Stelle des Aufrufs in den Objektcode genauso geschrieben, als dienten sie der Initialisierung von Daten. Es darf auch die Adresse eines Parameters oder einer automatischen Variablen plus oder minus einem konstanten Offset verwendet werden. Für diese Argumente wird der Offset der Variablen vom BP-Register gespeichert.

Die Anzahl der im Objektcode gespeicherten Bytes hängt, abgesehen von folgenden Einschränkungen, vom Typ des jeweiligen Arguments ab:

- eine vorzeichenbehaftete Integer-Konstante (zum Beispiel: 0x90) wird als ein Zeichen behandelt, wenn sie im Bereich zwischen 0 und 255 liegt;
- für die Adresse einer automatischen Variablen oder eines Parameters wird ein Byte geschrieben, wenn der Offset der Variablen vom BP-Register zwischen -128 und 127 liegt, andernfalls wird ein WORD (2 Bytes) geschrieben.

Ein einfaches Byte schreiben Sie so:

```
__emit__ (0x90);
```

Falls Sie ein Wort einfügen wollen, dessen Wert kleiner als 255 ist, modifizieren Sie den Typ einfach in **unsigned**:

```
__emit__ (0xB8, (unsigned)17);
```

oder

```
__emit__ (0xB8, 17u);
```

Zwei bzw. vier Byte lange Adressen können durch die Typumwandlung einer Adresse mit **void near*** bzw. **void far*** erzwungen werden.

Rückgabewert Keiner.

Beispiel
```
#include <dos.h>

int main(void)
{
/* Emit code that will generate a print screen
   via int 5 */
   __emit__(0xcd,0x05);
   return 0;
}
```

eof

Funktion Prüft, ob das Ende einer Datei erreicht ist.

Syntax
```
#include <io.h>
int eof(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *eof* prüft, ob der Dateizeiger der durch *handle* bezeichneten Datei am Dateiende steht.

Rückgabewert Wenn das Ende der Datei erreicht ist, liefert *eof* den Wert 1, ansonsten den Wert 0. Der Rückgabewert -1 zeigt einen Fehler an, die globale Variable *errno* bekommt in diesem Fall den Wert

EBADF Ungültiges Handle

Siehe auch *clearerr, feof, ferror, perror*

Beispiel
```
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "This is a test";
   char ch;

   /* create a file */
   handle = open("DUMMY.FIL",
```

```
                    O_CREAT | O_RDWR,
                    S_IREAD | S_IWRITE);

  /* write some data to the file */
  write(handle, msg, strlen(msg));

  /* seek to the beginning of the file */
  lseek(handle, 0L, SEEK_SET);

 /* reads chars from the file until it reaches EOF */
  do
  {
     read(handle, &ch, 1);
     printf("%c", ch);
  } while (!eof(handle));

  close(handle);
  return 0;
}
```

exec...

Funktion

Funktionen zum Laden und Starten eines anderen Programms.

Syntax

```
#include <process.h>
int execl(char *path, char *arg0, ...,
          *argn, NULL);
int execle(char *path, char *arg0, ...,
           *argn, NULL, char **env);
int execlp(char *path, char *arg0, ...,
           *argn, NULL);
int execlpe(char *path, char *arg0, ...,
            *argn, NULL, char **env);
int execv(char *path, char *argv[]);
int execve(char *path, char *argv[], char **env);
int execvp(char *path, char *argv[]);
int execvpe(char *path, char *argv[], char **env);
```

DOS	UNIX	Windows	ANSI C	nur C++
	x			

Beschreibung

Die Funktionen der *exec...*-Familie laden und starten andere Programme, die als *untergeordnete Prozesse* bezeichnet werden. Wenn ein *exec...*-Aufruf fehlerfrei ausgeführt werden konnte, überlagert der untergeordnete Prozeß (child process) seinen übergeordneten Prozeß (parent process). Im Hauptspeicher muß genügend Platz sein, um den untergeordneten Prozeß laden und ausführen zu können. Der Parameter *path* gibt den Dateinamen des *untergeordneten Prozesses* an – verwendet wird der Standard-Suchalgorithmus von DOS:

- wenn weder eine Namenserweiterung noch ein Punkt angegeben ist, wird zuerst nach Dateien ohne Namenserweiterung, dann nach .COM-Dateien und anschließend nach .EXE-Dateien gesucht;

- wenn eine Namenserweiterung oder ein Punkt angegeben ist, wird nur nach dem angegebenen Dateinamen gesucht.

Die einzelnen Funktionen der *exec...*-Familie unterscheiden sich durch die Namensendungen *l*, *v*, *p* und *e*, wobei jeder dieser Buchstaben für andere Fähigkeiten steht.
Es bedeuten:

p Die *exec*-Funktion sucht nicht nur innerhalb des aktuellen Verzeichnisses nach der Datei, sondern auch in sämtlichen in der DOS-Variablen PATH festgelegten Verzeichnissen, falls der Parameter *path* kein explizites Verzeichnis enthält.

l Die Zeiger (*arg0*, ..., *argn*) werden in Form einer Liste einzelner Argumente übergeben. Die Listenform wird normalerweise verwendet, wenn die Zahl der zu übergebenden Argumente bekannt und konstant ist.

v Die Zeiger werden als Zeiger-Array übergeben, das heißt als *argv*[*0*], *argv*[*1*] ...*argv*[*n*]. Diese Form wird normalerweise bei einer variablen Zahl von Argumenten verwendet.

e Erlaubt die Übergabe des Zeiger-Array *env* (d.h. eines eigenen *Environment* für den untergeordneten Prozeß). Über *exec...*-Varianten ohne »*e*« aufgerufene Prozesse verwenden dieselben Umgebungsparameter wie das aufrufende Programm.

Jede Variante der *exec...*-Familie enthält entweder ein *l* oder ein *v* im Namen, die weiteren Kombinationen mit *p* und *e* sind optional.
Zwei Beispiele dazu:

execl verwendet eine Liste von Argumenten, sucht nur das aktuelle Verzeichnis oder das Stammverzeichnis nach dem untergeordneten Prozeß ab und übergibt dann die *Environment*-Parameter, die auch für das momentan laufende Programm gültig sind;

execvpe übernimmt ein Zeiger-Array, führt nötigenfalls eine Suche nach dem untergeordneten Prozeß über die durch PATH gesetzten Suchwege aus und übergibt *Environment*-Parameter, die vom Programmierer für diesen Prozeß definiert worden sind.

Sämtliche *exec...*-Funktionen müssen zumindest ein Argument an den untergeordneten Prozeß übergeben, nämlich *arg0* bzw. *argv*[*0*]. Dieses Argument enthält per Konvention eine Kopie von *path*. Die Übergabe anderer Daten als erstem

Parameter widerspricht nur den Gepflogenheiten, ist aber kein Fehler.

Der Wert von *arg0* bzw. *argv[0]* ist nur unter den DOS-Versionen ab 3.0 für den untergeordneten Prozeß verfügbar. Bei Verwendung der »*l*«-Funktionen zeigt *arg0* also normalerweise auf denselben String wie *path*.

arg1,...,argn zeigen auf weitere Strings, die Argumentliste wird durch ein obligatorisches NULL beendet.

Die »*e*«-Funktionen erwarten das Argument *env* als Zeiger-Array. Jedes Element dieses Array ist ein nullterminierter String der Form

```
E_Variable=Wert
```

wobei *E_Variable* der Name einer Environment-Variablen ist und *Wert* der String, auf den *E_Variable* gesetzt wird.

Das letzte Element des Array muß einen Zeiger mit dem Wert NULL enthalten. Dadurch wird das Listenende gekennzeichnet. Wenn *env[0]* den Wert NULL hat, verwendet der untergeordnete Prozeß die *Environment*-Tabelle seines übergeordneten Prozesses.

Die Gesamtlänge sämtlicher als *arg* übergebenen Argumente darf 128 Byte nicht überschreiten (inklusive der automatisch eingesetzten Leerzeichen zur Trennung einzelner Parameter). Nullzeichen und *Environment*-Parameter werden bei dieser Zählung nicht berücksichtigt.

exec... schließt keine Dateien – sämtliche durch den übergeordneten Prozeß geöffneten Dateien stehen auch dem untergeordneten Prozeß zur Verfügung.

Rückgabewert

Bei fehlerfreier Ausführung kehrt eine *exec...*-Funktion überhaupt nicht zum Programm zurück. Konnte die *exec...*-Funktion dagegen nicht ausgeführt werden, dann wird -1 zurückgeliefert, und die globale Variable *errno* bekommt einen der folgenden Werte:

E2BIG	Argumentliste zu lang
EACCES	Zugriff nicht erlaubt
EMFILE	Zu viele offene Dateien
ENOENT	Suchweg/Dateiname nicht gefunden
ENOEXEC	*exec*-Formatfehler
ENOMEM	Nicht genug Platz im Hauptspeicher

Siehe auch

abort, *atexit*, *_exit*, *exit*, *_fpreset*, *searchpath*, *spawn...*, *system*

Beispiel

Für jede Funktion der exec...-Familie ist ein eigenes Beispiel bereitgestellt.

```
/* EXECL() EXAMPLE */

#include <stdio.h>
#include <errno.h>
```

```
#include <dos.h>
int main(int argc, char *argv[])
{
  int loop;
  printf("%s running...\n\n", argv[0]);
  if (argc == 1) {
    /* check for only one command-line parameter */
    printf("%s calling itself again...\n", argv[0]);
    execl(argv[0], argv[0], "ONE", "TWO", "THREE",
          NULL);
    perror("EXEC:");
    exit(1);
  }
  printf("%s called with arguments:\n", argv[0]);
  for (loop = 1; loop <= argc; loop++)
  puts(argv[loop]);
       /* display all command-line parameters */
  return 0;
}
```

```
/* EXECLE EXAMPLE */

#include <process.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[], char *env[])
{
  int loop;
  char *new_env[] = { "TESTING", NULL };
  printf("%s running...\n\n", argv[0]);
  if (argc == 1) {   /* check for only one
                        command-line parameter */
    printf("%s calling itself again...\n", argv[0]);
    execle(argv[0], argv[0], "ONE", "TWO", "THREE",
           NULL, new_env);
    perror("EXEC:");
    exit(1);
  }
  printf("%s called with arguments:\n", argv[0]);
  for (loop = 1; loop <= argc; loop++)
    puts(argv[loop]);  /* display all command-line
                          parameters */

  /* display the first environment parameter */
  printf("value of env[0]: %s\n",env[0]);
  return 0;
}
```

```
/* EXECLPE() EXAMPLE */

#include <process.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[], char **envp)
{
  int i;
  printf("Command line arguments:\n");
  for (i=0; i < argc; ++i)
    printf("[%2d] %s\n", i, argv[i]);
  printf("About to exec child with arg1 arg2 ...\n");
  execlpe("CHILD.EXE", "CHILD.EXE", "arg1", "arg2",
          NULL, envp);
  perror("exec error");
  exit(1);
  return 0;
}
```

```
/* EXECLPE() EXAMPLE */

#include <process.h>
#include <stdio.h>
#include <errno.h>

int main( int argc, char *argv[])
{
   int i;

   printf("Command line arguments:\n");

   for (i=0; i < argc; ++i)
      printf("[%2d] %s\n", i, argv[i]);

   printf("About to exec child with arg1 arg2 ...\n");
   execlp("CHILD.EXE", "CHILD.EXE", "arg1", "arg2",
           NULL);

   perror("exec error");
   exit(1);

   return 0;
}
```

```
/* EXECV() EXAMPLE */

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
  int i;
```

```
  printf("Command line arguments:\n");
  for (i=0; i<argc; i++)
    printf("[%2d] : %s\n", i, argv[i]);
  printf("About to exec child with arg1 arg2 ...\n");
  execv("CHILD.EXE", argv);
  perror("exec error");
  exit(1);
}

/* EXECVE() EXAMPLE */

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[], char **envp)
{
  int i;
  printf("Command line arguments:\n");
  for (i=0; i<argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);
  printf("About to exec child with arg1 arg2 ...\n");
  execve("CHILD.EXE", argv, envp);
  perror("exec error");
  exit(1);
}

/* EXECVP() EXAMPLE */

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
  int i;
  printf("Command line arguments:\n");
  for (i=0; i<argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);
  printf("About to exec child with arg1 arg2 ...\n");
  execvp("CHILD.EXE", argv);
  perror("exec error");
  exit(1);
}

/* EXECVPE() EXAMPLE */

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[], char **envp)
{
```

```
  int i;
  printf("Command line arguments:\n");
  for (i=0; i<argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);
  printf("About to exec child with arg1 arg2 ...\n");
  execvpe("CHILD.EXE", argv, envp);
  perror("exec error");
  exit(1);
}
```

_exit

Funktion: Beendet das laufende Programm.

Syntax:

```
#include <stdlib.h>
void _exit(int status);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung: *_exit* beendet das Programm, ohne Dateien zu schließen, Ausgabepuffer zu leeren oder Exit-Funktionen aufzurufen.

_exit beendet das Programm mit dem als *status* angegebenen Exit-Code, der vom aufrufenden Prozeß geprüft werden kann. Der Wert 0 steht normalerweise für eine fehlerfreie Ausführung, Werte ungleich Null signalisieren Fehler.

Rückgabewert: Keiner.

Siehe auch: *abort, atexit, exec..., exit, spawn...*

Beispiel:

```
#include   <stdlib.h>
#include   <stdio.h>

void done(void);

int main(void)
{

   atexit(done);
   _exit(0);
   return 0;
}

void done()
{
   printf("hello\n");
}
```

exit

Funktion Beendet das laufende Programm.

Syntax
```
#include <stdlib.h>
void exit(int status);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *exit* beendet das aufrufende Programm. Vor dem Rücksprung werden sämtliche Dateipuffer geschrieben, offene Dateien geschlossen und (mit *atexit*) festgelegte Exit-Funktionen aufgerufen.

Die Funktion beendet das Programm mit dem als *status* angegebenen Exit-Code, der vom aufrufenden Prozeß geprüft werden kann.

Der Wert 0 steht normalerweise für eine fehlerfreie Ausführung, Werte ungleich Null signalisieren Fehler.

Folgende Konstanten stehen für *status* zur Verfügung:

EXIT_SUCCESS	Normales Programmende.
EXIT_FAILURE	Fehlerhaftes Programmende – Signal an das Betriebssystem, daß das Programm mit einem Fehler endete.

Rückgabewert Keiner.

Siehe auch *abort*, *atexit*, *exec...*, *_exit*, *keep*, *signal*, *spawn...*

Beispiel
```
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

int main(void)
{
   int status;

   printf("Enter either 1 or 2\n");
   status = getch();
   /* Sets DOS errorlevel  */
   exit(status - '0');

/* Note: this line is never reached */
   return 0;
}
```

exp, expl

Funktion Berechnet die Exponentialfunktion e^x.

Syntax

reelle Version:

```
#include <math.h>
double exp(double x);
long double expl(long double x);
```

komplexe Version:

```
#include <complex.h>
complex exp(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*expl*
x	x	x	x		*reelles exp*
x		x		x	*komplexes exp*

Beschreibung *exp* berechnet die Exponentialfunktion e^x.

expl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Die komplexe Exponentialfunktion ist definiert durch:

$$exp(x + y\,i) = exp(x)\,(cos(y) + i\,sin(y))$$

Rückgabewert *exp* liefert e^x zurück. Falls die Argumente zu unkalkulierbaren Resultaten führen, werden die folgenden Fehlerbedingungen erzeugt: Bei einem Überlauf liefert *exp* HUGE_VAL und *expl* _LHUGE_VAL. Die globale Variable *errno* wird in diesen Fällen auf folgenden Wert gesetzt:

ERANGE Ergebnis außerhalb des Rechenbereichs

Bei einem Unterlauf liefern *exp* und *expl* 0.0 zurück, die globale Variable *errno* wird nicht verändert.

Die Behandlung von Fehlern dieser Funktionen kann über die Routinen *matherr* und *_matherrl* modifiziert werden.

Siehe auch *frexp, ldexp, log, log10, matherr, pow, pow10, sqrt*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 4.0;

   result = exp(x);
   printf("'e' raised to the power \
          of %lf (e ^ %lf) = %lf\n",
          x, x, result);

   return 0;
}
```

fabs, fabsl

Funktion Berechnet den absoluten Wert einer Gleitkommazahl.

Syntax
```
#include <math.h>
double fabs(double x);
long double fabsl(long double x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*fabs*
x		x			*fabsl*

Beschreibung *fabs* berechnet den absoluten Betrag des **double**-Arguments *x*. *fabsl* ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double**.

Rückgabewert *fabs* und *fabsl* liefern den absoluten Betrag von *x* zurück.

Siehe auch *abs, cabs, labs*

Beispiel
```
#include <stdio.h>
#include <math.h>

int main(void)
{
   float  number = -1234.0;

   printf("number: %f  absolute value: %f\n",
          number, fabs(number));
   return 0;
}
```

farcalloc

Funktion Belegt Speicherplatz auf dem FAR-Heap.

Syntax
```
#include <alloc.h>
void far *farcalloc(unsigned long nunits,
                    unsigned long unitsz);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *farcalloc* belegt Platz auf dem FAR-Heap für ein Array mit *nunits* Elementen, deren Größe jeweils *unitsz* Bytes beträgt.

Bei der Arbeit mit dem FAR-Heap« sind einige sonst vorhandene Grenzen aufgehoben:

- der gesamte vom Computer bzw. von DOS bereitgestellte RAM-Speicherplatz kann belegt werden;

- die Größe einzelner Blöcke kann 64 KByte überschreiten;
- für Zugriffe auf die Blöcke müssen grundsätzlich **far**-Zeiger (oder **huge**-Zeiger, wenn die Blöcke größer als 64 KByte sind) benutzt werden.

Bei Verwendung der Speichermodelle *compact*, *large* und *huge* verhält sich *farcalloc* ähnlich wie *calloc*, die Funktionen sind aber nicht identisch: die Parameter von *calloc* sind vom Typ **unsigned**, diejenigen von *farcalloc* dagegen vom Typ **unsigned long**.

Im Modell *tiny* kann *farcalloc* nicht verwendet werden.

Rückgabewert *farcalloc* liefert einen **far**-Zeiger auf die Startadresse des neu belegten Speicherblocks zurück. Wenn nicht ausreichend Platz zur Verfügung steht, wird der Zeigerwert NULL zurückgeliefert.

Siehe auch *calloc*, *farcoreleft*, *farfree*, *farmalloc*, *malloc*

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
  char far *fptr;
  char *str = "Hello";

   /* allocate memory for the far pointer */
   fptr = (char far *) farcalloc(10, sizeof(char));

    /* copy "Hello" into allocated memory */
    /*
      Note: movedata is used because you might be in
      a small data model, in which case a normal
      string copy routine can not be used since it
      assumes the pointer size is near.
   */
   movedata(FP_SEG(str), FP_OFF(str),
            FP_SEG(fptr), FP_OFF(fptr),
            strlen(str));

   /* display string (note the F modifier) */
   printf("Far string is: %Fs\n", fptr);

   /* free the memory */
   farfree(fptr);

   return 0;
}
```

farcoreleft

Funktion Ermittelt den unbenutzten Speicherplatz auf dem FAR-Heap.

Syntax
```
#include <alloc.h>
unsigned long farcoreleft(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *farcoreleft* ermittelt, wieviel Speicherplatz auf dem FAR-Heap oberhalb des obersten belegten Blocks zur Verfügung steht.
Im Modell *tiny* kann *farcoreleft* nicht verwendet werden.

Rückgabewert *farcoreleft* liefert die Größe des noch verfügbaren Speicherplatzes zurück. Ermittelt wird die Differenz zwischen der Endadresse des höchsten momentan belegten Blocks und dem Ende des verfügbaren Speichers.

Siehe auch *coreleft*, *farcalloc*, *farmalloc*

Beispiel
```
#include <stdio.h>
#include <alloc.h>

int main(void)
{
 printf("The difference between the highest "
        "allocated block in the far\n");
 printf("heap and the top of the far heap is: "
        "%lu bytes\n", farcoreleft());

   return 0;
}
```

farfree

Funktion Gibt Speicher auf dem FAR-Heap frei.

Syntax
```
#include <alloc.h>
void farfree(void far *block);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *farfree* gibt einen zuvor belegten Speicherblock auf dem FAR-Heap wieder frei. Im Modell *Tiny* kann *farfree* nicht verwendet werden. In den Modellen *Small* und *Medium* sind der Heap und der FAR-Heap zwei völlig voneinander getrennte Speicherbereiche – durch *farmalloc* belegte Blöcke können nicht mit *free*

freigegeben werden, durch *malloc* belegte Blöcke nicht mit *farfree*.

Rückgabewert Keiner.

Siehe auch *farcalloc, farmalloc*

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char far *fptr;
   char *str = "Hello";

   /* allocate memory for the far pointer */
   fptr = (char far *) farcalloc(10, sizeof(char));

   /* copy "Hello" into allocated memory */
   /*
      Note: movedata is used because you might be in
      a small data model, in which case a normal
      string copy routine can't be used since it
      assumes the pointer size is near.
   */
   movedata(FP_SEG(str), FP_OFF(str),
            FP_SEG(fptr), FP_OFF(fptr),
            strlen(str));

   /* display string (note the F modifier) */
   printf("Far string is: %Fs\n", fptr);

   /* free the memory */
   farfree(fptr);

   return 0;
}
```

farheapcheck

Funktion Überprüft den FAR-Heap.

Syntax

```
#include <alloc.h>
int farheapcheck(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *farheapcheck* überprüft jeden Block des FAR-Heap auf seine Zeiger, Größe und weitere wichtige Attribute.

Rückgabewert Bei fehlerfreier Ausführung wird ein Wert größer als Null, im Fall eines Fehlers ein Wert kleiner als Null zurückgeliefert. Im einzelnen können für die Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen FAR-Heap (Wert 1)
_HEAPOK	Kein Fehler gefunden (Wert 2)
_HEAPCORRUPT	Der FAR-Heap ist ungültig (Wert -1).

Siehe auch *heapcheck*

Beispiel

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
char far *array[ NUM_PTRS ];
int i;

    for( i = 0; i < NUM_PTRS; i++ )
     array[ i ] = (char far *) farmalloc( NUM_BYTES );

    for( i = 0; i < NUM_PTRS; i += 2 )
         farfree( array[ i ] );

    if( farheapcheck() == _HEAPCORRUPT )
        printf( "Heap is corrupted.\n" );
    else
        printf( "Heap is OK.\n" );

    return 0;
}
```

farheapcheckfree

Funktion Überprüft die freien Blöcke auf dem FAR-Heap auf einen konstanten Wert.

Syntax

```
#include <alloc.h>
int farheapcheckfree(unsigned int fillvalue);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *farheapcheckfree* überprüft die freien Blöcke des FAR-Heap auf einen konstanten Füllwert.

Rückgabewert Bei fehlerfreier Ausführung ist der Rückgabewert größer als Null, im Fall eines Fehlers kleiner als Null.

Im einzelnen können für die Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen FAR-Heap (Wert 1)
_HEAPOK	Kein Fehler gefunden (Wert 2)
_HEAPCORRUPT	Der FAR-Heap ist ungültig (Wert -1)
_BADVALUE	Ein anderer Wert als der Füllwert wurde gefunden (Wert -3)

Siehe auch *farheapfillfree, heapcheckfree*

Beispiel

```
#include <mem.h>
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
  char far *array[NUM_PTRS];
  int i;
  int j;
  int res;

  for (i = 0; i < NUM_PTRS; i++)
    if ((array[i] = (char far *) farmalloc(NUM_BYTES))
        == NULL)
    {
      printf("No memory for allocation\n");
      return 1;
    }

  for (i = 0; i < NUM_PTRS; i += 2)
    farfree(array[i]);

  if(farheapfillfree(1) < 0)
  {
    printf("Heap corrupted.\n");
    return 1;
  }

  for (i = 1; i < NUM_PTRS; i += 2)
    for (j = 0; j < NUM_BYTES; j++)
      array[i][j] = 0;

  res = farheapcheckfree(1);
  if (res < 0)
    switch(res)
    {
      case _HEAPCORRUPT:
```

```
        printf("Heap corrupted.\n");
        return 1;
      case _BADVALUE:
        printf("Bad value in free space.\n");
        return 1;
      default:
        printf("Unknown error.\n");
        return 1;
    }
  printf("Test successful.\n");
  return 0;
}
```

farheapchecknode

Funktion: Überprüft einen einzelnen Knoten auf dem FAR-Heap.

Syntax:

```
#include <alloc.h>
int farheapchecknode(void *node);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung: *farheapchecknode* überprüft die einzelnen Knoten auf dem FAR-Heap. Wird *farheapchecknode* nach der Freigabe eines Knotens mit einem Zeiger auf diesen Block aufgerufen, kann _BADNODE anstelle des erwarteten _FREEENTRY zurückgeliefert werden. Der Grund liegt darin, daß benachbarte freie Blöcke auf dem FAR-Heap zusammengefaßt wurden und der betreffende Block deshalb nicht mehr existiert.

Rückgabewert: Bei fehlerfreier Ausführung ist der Rückgabewert größer als Null, im Fall eines Fehlers kleiner als Null. Im einzelnen können für die Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen FAR-Heap« (Wert 1)
_HEAPCORRUPT	Der FAR-Heap ist ungültig (Wert -1)
_BADNODE	Der Knoten wurde nicht gefunden (Wert -2)
_FREEENTRY	Der Knoten ist ein freier Block (Wert 3)
_USEDENTRY	Der Knoten ist ein belegter Block (Wert 4)

Siehe auch: *heapchecknode*

Beispiel:

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16
int main(void)
```

```
{
   char far *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
     array[ i ] = (char far *) farmalloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      farfree( array[ i ] );

   for( i = 0; i < NUM_PTRS; i++ )
   {
      printf( "Node %2d ", i );
      switch( farheapchecknode( array[ i ] ))
      {
         case _HEAPEMPTY:
            printf("No heap.\n" );
            break;
         case _HEAPCORRUPT:
            printf("Heap corrupt.\n" );
            break;
         case _BADNODE:
            printf("Bad node.\n" );
            break;
         case _FREEENTRY:
            printf("Free entry.\n" );
            break;
         case _USEDENTRY:
            printf("Used entry.\n" );
            break;
         default:
            printf("Unknown return code.\n");
            break;
         }
      }

   return 0;
}
```

farheapfillfree

Funktion

Füllt die freien Blöcke des FAR-Heap mit einem konstanten Wert.

Syntax

```
#include <alloc.h>
int farheapfillfree(unsigned int fillvalue);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *farheapfillfree* füllt die freien Blöcke des FAR-Heap mit einem konstanten Wert.

Rückgabewert Bei fehlerfreier Ausführung ist der Rückgabewert größer als Null, im Fall eines Fehlers kleiner als Null. Im einzelnen können für die Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen FAR-Heap« (Wert 1)
_HEAPOK	Kein Fehler gefunden (Wert 2)
_HEAPCORRUPT	Der FAR-Heap ist ungültig (Wert -1)

Siehe auch *farheapcheckfree, heapfillfree*

Beispiel

```
#include <mem.h>
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char far *array[NUM_PTRS];
   int i;
   int j;
   int res;

   for (i = 0; i < NUM_PTRS; i++)
      if ((array[i] = (char far *)
          farmalloc(NUM_BYTES)) == NULL)
      {
         printf("No memory for allocation\n");
         return 1;
      }

   for (i = 0; i < NUM_PTRS; i += 2)
      farfree(array[i]);

   if(farheapfillfree(1) < 0)
   {
      printf("Heap corrupted.\n");
      return 1;
   }

   for (i = 1; i < NUM_PTRS; i += 2)
      for (j = 0; j < NUM_BYTES; j++)
         array[i][j] = 0;

   res = farheapcheckfree(1);
   if (res < 0)
      switch(res)
      {
         case _HEAPCORRUPT:
```

```
            printf("Heap corrupted.\n");
            return 1;
         case _BADVALUE:
            printf("Bad value in free space.\n");
            return 1;
         default:
            printf("Unknown error.\n");
            return 1;
      }

   printf("Test successful.\n");
   return 0;
}
```

farheapwalk

Funktion Durchwandert den FAR-Heap Knoten für Knoten.

Syntax
```
#include <alloc.h>
int farheapwalk(struct farheapinfo *hi);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *farheapwalk* durchwandert den FAR-Heap Knoten für Knoten. Die Funktion geht davon aus, daß der FAR-Heap gültig ist (überprüfen Sie dies vorher mit *farheapcheck*). Mit dem letzten Block wird _HEAPOK zurückgeliefert. Bei einem erneuten Aufruf von *farheapwalk* liefert die Funktion _HEAPEND.

farheapwalk erhält einen Zeiger auf eine Struktur vom Typ *farheapinfo* (definiert in alloc.h). Setzen Sie das Feld *hi.ptr* beim ersten Aufruf der Funktion auf Null. Nach der Rückkehr enthält *hi.ptr* die Adresse des ersten Blocks, während *hi.size* die Größe des Blocks in Bytes angibt. Ist der Block gerade belegt, wird das Flag *hi.in_use* gesetzt.

Rückgabewert Folgende Werte können zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen FAR-Heap (Wert 1)
_HEAPOK	Der *farheapinfo*-Block enthält gültige Daten (Wert 2)
_HEAPEND	das Ende des FAR-Heap wurde erreicht (Wert 5)

Siehe auch *heapwalk*

Beispiel
```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16
```

```
int main( void )
{
   struct farheapinfo hi;
   char far *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
     array[ i ] = (char far *) farmalloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      farfree( array[ i ] );

   hi.ptr = NULL;
   printf( "   Size   Status\n" );
   printf( "   ----   ------\n" );
   while( farheapwalk( &hi ) == _HEAPOK )
      printf( "%7u    %s\n", hi.size, hi.in_use ?
             "used" : "free" );
}
```

farmalloc

Funktion Belegt Speicher auf dem FAR-Heap.

Syntax
```
#include <alloc.h>
void far *farmalloc(unsigned long nbytes);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung *farmalloc* belegt einen Speicherblock von *nbytes* Größe auf dem FAR-Heap. Der Inhalt dieses Blocks ist zum Zeitpunkt der Belegung undefiniert und bleibt es auch (wie *malloc* nimmt *farmalloc* keine Initialisierung vor).

Bei der Arbeit mit dem »far Heap« sind einige sonst vorhandene Grenzen aufgehoben:

- der gesamte vom Computer bzw. von DOS bereitgestellte RAM-Speicherplatz kann belegt werden;
- die Größe einzelner Blöcke kann 64 KByte überschreiten;
- für Zugriffe auf die Blöcke müssen grundsätzlich **far**-Zeiger benutzt werden.

Bei Verwendung der Speichermodelle *compact, large* und *huge* verhält sich *farmalloc* ähnlich wie die Funktion *malloc* – identisch sind sie aber nicht: die Parameter von *malloc* sind vom Typ **unsigned** , die von *farmalloc* dagegen vom Typ **unsigned long** .

Im Modell *tiny* kann *farmalloc* nicht verwendet werden.

Rückgabewert *farmalloc* liefert einen **far**-Zeiger auf die Startadresse des neu belegten Speicherblocks zurück. Wenn nicht ausreichend Platz zur Verfügung steht, wird der Zeigerwert NULL zurückgeliefert.

Siehe auch *farcalloc, farcoreleft, farfree, farrealloc, malloc*

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char far *fptr;
   char *str = "Hello";

   /* allocate memory for the far pointer */
   fptr = (char far *) farmalloc(10);

   /* copy "Hello" into allocated memory */
   /*
      Note: movedata is used because we might be in
      a small data model, in which case a normal
      string copy routine can not be used since it
      assumes the pointer size is near.
   */
   movedata(FP_SEG(str), FP_OFF(str),
            FP_SEG(fptr), FP_OFF(fptr),
            strlen(str) + 1);

   /* display string (note the F modifier) */
   printf("Far string is: %Fs\n", fptr);

   /* free the memory */
   farfree(fptr);

   return 0;
}
```

farrealloc

Funktion Vergrößert/verkleinert einen auf dem FAR-Heap belegten Speicherbereich.

Syntax

```
#include <alloc.h>
void far *farrealloc(void far *oldblock,
                     unsigned long nbytes);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *farrealloc* verändert die Größe eines bereits belegten Blocks auf *nbytes*, wobei der Blockinhalt nötigenfalls in einen anderen Speicherbereich kopiert wird.

Bei der Arbeit mit dem FAR-Heap sind einige sonst vorhandene Grenzen aufgehoben:

- der gesamte vom Computer bzw. von DOS bereitgestellte RAM-Speicherplatz kann belegt werden;
- die Größe einzelner Blöcke kann 64 KByte überschreiten;
- für Zugriffe auf die Blöcke müssen grundsätzlich **far**-Zeiger benutzt werden.

Im Modell *tiny* kann *farrealloc* nicht verwendet werden.

Rückgabewert *farrealloc* liefert einen **far**-Zeiger auf die Startadresse des veränderten Speicherblocks zurück, die sich von der alten Startadresse unterscheiden kann. Falls nicht ausreichend Platz zur Verfügung steht, wird der Wert NULL geliefert.

Siehe auch *farmalloc, realloc*

Beispiel

```
#include <stdio.h>
#include <alloc.h>

int main(void)
{
   char far *fptr;
   char far *newptr;

   fptr = (char far *) farmalloc(16);
   printf("First address: %Fp\n", fptr);

   /*  we use a second pointer, newptr, so that in
       the case of farrealloc() returning NULL, our
       original pointer is not set to NULL.        */

   newptr = (char far *) farrealloc(fptr,64);
   printf("New address  : %Fp\n", newptr);
   if (newptr != NULL)
      farfree(newptr);
   return 0;
}
```

fclose

Funktion Schließt einen Stream.

Syntax

```
#include <stdio.h>
int fclose(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fclose* schließt den durch *stream* angegebenen Stream, wobei zuvor ein entsprechender Aufruf von *fflush* erfolgt. Durch das System belegte Puffer werden durch *fclose* freigegeben.

Puffer, die mit *setbuf* oder *setvbuf* zugeordnet wurden, werden nicht automatisch freigegeben (es sei denn, für den Pufferzeiger von *setvbuf* wurde NULL übergeben).

Rückgabewert *fclose* liefert bei fehlerfreier Ausführung den Wert 0, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch *close*, *fcloseall*, *fdopen*, *fflush*, *flushall*, *fopen*, *freopen*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   FILE *fp;
   char buf[11] = "0123456789";

   /* create a file containing 10 bytes */
   fp = fopen("DUMMY.FIL", "w");
   fwrite(&buf, strlen(buf), 1, fp);

   /* close the file */
   fclose(fp);
   return 0;
}
```

fcloseall

Funktion Schließt offene Streams.

Syntax

```
#include <stdio.h>
int fcloseall(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *fcloseall* schließt sämtliche offenen Streams außer *stdin*, *stdout*, *stderr*, *stdprn* und *stdaux*.

Rückgabewert *fcloseall* liefert bei fehlerfreier Ausführung die Anzahl der geschlossenen Streams zurück, im Fall eines Fehlers den Wert EOF.

Siehe auch *fclose*, *fdopen*, *flushall*, *fopen*, *freopen*

Beispiel

```
#include <stdio.h>

int main(void)
{
   int streams_closed;

   /* open two streams */
   fopen("DUMMY.ONE", "w");
   fopen("DUMMY.TWO", "w");

   /* close the open streams */
   streams_closed = fcloseall();

   if (streams_closed == EOF)
      /* issue an error message */
      perror("Error");
   else
      /* print result of fcloseall() function */
      printf("%d streams were closed.\n",
             streams_closed);

   return 0;
}
```

fcvt

Funktion

Konvertiert eine Gleitkommazahl in einen String.

Syntax

```
#include <stdlib.h>
char *fcvt(double value, int ndig, int *dec,
           int *sign);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

fcvt konvertiert *value* in einen nullterminierten String mit *ndig* Ziffern rechts vom Dezimalpunkt – beginnend mit der höchstwertigen Ziffer (links) – und liefert einen Zeiger auf diesen String zurück. Der String enthält ausschließlich Ziffern: über *dec* wird die Position des Dezimalpunkts relativ zum Beginn des Strings zurückgeliefert (negative Werte bedeuten hier: »Dezimalpunkt links vom Stringanfang«). Der String selbst enthält keinen Dezimalpunkt.

Wenn *value* negativ ist, dann setzt *fcvt sign* auf einen Wert ungleich Null (für positive Werte von *value* ist *sign* $== 0$). Die letzte Dezimalstelle des zurückgelieferten Strings ist gerundet.

Die Ziffern werden auf *ndig* Stellen gerundet.

Rückgabewert

fcvt liefert einen Zeiger zu einem statischen String zurück, dessen Inhalt durch jeden Aufruf dieser Funktion (oder von *ecvt*) neu gesetzt wird.

Siehe auch *ecvt, gcvt, sprintf*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
  char *str;
  double num;
  int dec, sign, ndig = 5;

  /* a regular number */
  num = 9.876;
  str = fcvt(num, ndig, &dec, &sign);
  printf("string = %10s decimal place = %d "
         "sign = %d\n", str, dec, sign);

  /* a negative number */
  num = -123.45;
  str = fcvt(num, ndig, &dec, &sign);
  printf("string = %10s decimal place = %d "
         "sign = %d\n", str, dec, sign);

  /* scientific notation */
  num = 0.678e5;
  str = fcvt(num, ndig, &dec, &sign);
  printf("string = %10s  decimal place= %d  "
         "sign = %d\n", str, dec, sign);
  return 0;
}
```

fdopen

Funktion Ordnet einem Datei-Handle einen Stream zu.

Syntax

```
#include <stdio.h>
FILE *fdopen(int handle, char *type);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *fdopen* ordnet einen Stream einem Datei-Handle zu, das von *creat*, *dup*, *dup2* oder *open* erhalten wurde. Der Typ des Streams muß zum Modus des offenen *handle* passen. Der Parameter *type* kann einen der folgenden Werte annehmen:

r Öffnung ausschließlich für Leseoperationen.

w Erzeugung einer Datei für Schreiboperationen.

a Öffnung für Schreiboperationen. Falls die Datei bereits existiert, werden zusätzliche Informationen an das aktuelle Da-

teiende angefügt, ansonsten wird sie für Schreiboperationen erzeugt.

r+ Öffnung einer Datei für Lese- und Schreiboperationen – die Datei muß bereits existieren.

w+ Erzeugung einer neuen Datei für Lese- und Schreiboperationen.

a+ Öffnung einer Datei zum Lesen und »Anhängen« neuer Daten durch Schreiboperationen. Falls die Datei bereits existiert, bleibt ihr alter Inhalt erhalten – falls nicht, wird sie automatisch erzeugt.

Bei der Öffnung kann zusätzlich explizit angegeben werden, ob die Datei im Modus *text* (t) oder *binary* (b) bearbeitet werden soll. Die Angabe geschieht durch ein einfaches Anhängen des jeweiligen Buchstabens (also *wb*, *w+b*, *rt*, *a+t* etc.). Wenn sie fehlt, wird der jeweilige Modus durch die globale Variable *_fmode* bestimmt, deren Wert über die direkte Zuweisung einer der in fcntl.h definierten Konstanten O_TEXT oder O_BINARY festgelegt werden kann.

Mit *r+* und *w+* eröffnete Dateien können sowohl gelesen als auch beschrieben werden – zwischen einem Wechsel von »Schreiben« nach »Lesen« muß allerdings ein Aufruf von *fseek* oder *rewind* stattfinden. Zwischen einem Wechsel von »Lesen« nach »Schreiben« muß ebenfalls ein Aufruf von *fseek* oder *rewind* stattfinden, es sei denn, die Leseoperation hat das Dateiende erreicht.

Rückgabewert *fdopen* liefert bei fehlerfreier Ausführung einen Zeiger auf den neu geöffneten Stream zurück, im Fall eines Fehlers den Wert NULL.

Siehe auch *fclose*, *fopen*, *freopen*, *open*

Beispiel

```
#include <sys\stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   FILE *stream;

   /* open a file */
   handle = open("DUMMY.FIL", O_CREAT,
                 S_IREAD | S_IWRITE);

   /* now turn the handle into a stream */
   stream = fdopen(handle, "w");

   if (stream == NULL)
```

```
      printf("fdopen failed\n");
   else
   {
      fprintf(stream, "Hello world\n");
      fclose(stream);
   }
   return 0;
}
```

feof

Funktion Prüft, ob das Ende eines Stream erreicht ist.

Syntax

```
#include <stdio.h>
int feof(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung Das Makro *feof* prüft das Dateiende-Flag des über *stream* angegebenen Stream. Dieses Flag wird beim Erreichen des Dateiendes automatisch gesetzt. Leseoperationen liefern dann dieses Flag solange, bis *rewind* aufgerufen oder die Datei geschlossen wird. Vor jeder Eingabeoperation in diese Datei wird das Flag gelöscht.

Rückgabewert *feof* liefert einen Wert ungleich 0 zurück, wenn das Dateiende-Flag des angegebenen *stream* gesetzt ist, ansonsten ist der Rückgabewert 0.

Siehe auch *clearerr, eof, ferror, perror*

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *stream;

   /* open a file for reading */
   stream = fopen("DUMMY.FIL", "r");

   /* read a character from the file */
   fgetc(stream);

   /* check for EOF */
   if (feof(stream))
      printf("We have reached end-of-file\n");

   /* close the file */
   fclose(stream);
   return 0;
}
```

ferror

Funktion — Prüft Fehlerbedingungen bei der Bearbeitung eines Stream.

Syntax

```
#include <stdio.h>
int ferror(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *ferror* ist ein Makro, das den über *stream* angegebenen Stream auf Schreib- und Lesefehler prüft. Das Fehler-Flag eines Stream bleibt nach einem Fehler solange gesetzt, bis entweder *clearerr* oder *rewind* aufgerufen oder der Stream geschlossen wird.

Rückgabewert — *ferror* liefert einen Wert ungleich Null zurück, wenn das Fehler-Flag des angegebenen *stream* gesetzt ist, sonst den Wert 0.

Siehe auch — *clearerr*, *eof*, *feof*, *fopen*, *gets*, *perror*

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *stream;

   /* open a file for writing */
   stream = fopen("DUMMY.FIL", "w");

  /* force an error condition by attempting to read */
   (void) getc(stream);

   if (ferror(stream))
       /* test for an error on the stream */
   {
      /* display an error message */
      printf("Error reading from DUMMY.FIL\n");

      /* reset the error and EOF indicators */
      clearerr(stream);
   }

   fclose(stream);
   return 0;
}
```

fflush

Funktion — Schreibt den Ausgabepuffer eines Stream in die zugeordnete Datei.

Syntax

```
#include <stdio.h>
int fflush(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fflush* erzwingt bei gepufferten Ausgabe-Streams ein Schreiben eventueller Pufferinhalte, sorgt also dafür, daß ausgegebene Daten in die zugehörige Datei geschrieben werden.

Der durch *stream* bezeichnete Stream bleibt offen. Auf ungepufferte Streams hat *fflush* keine Auswirkung.

Rückgabewert Bei fehlerfreier Ausführung liefert *fflush* den Wert 0, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch *fclose, flushall, setbuf, setvbuf*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

void flush(FILE *stream);

int main(void)
{
   FILE *stream;
   char msg[] = "This is a test";

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");

   /* write some data to the file */
   fwrite(msg, strlen(msg), 1, stream);

   clrscr();
   printf("Press any key to flush DUMMY.FIL:");
   getch();

  /* flush the data to DUMMY.FIL without closing it */
   flush(stream);

   printf("\nFile was flushed, "
          "Press any key to quit:");
   getch();
   return 0;
}

void flush(FILE *stream)
{
     int duphandle;

     /* flush the stream's internal buffer */
     fflush(stream);

     /* make a duplicate file handle */
     duphandle = dup(fileno(stream));
```

```
    /* close the duplicate handle */
    /* to flush the DOS buffer */
    close(duphandle);
}
```

fgetc

Funktion — Liest ein Zeichen aus einem Stream.

Syntax

```
#include <stdio.h>
int fgetc(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *fgetc* liest das jeweils nächste Zeichen aus dem Eingabe-Stream *stream*.

Rückgabewert — *fgetc* konvertiert bei fehlerfreier Ausführung das gelesene Zeichen in einen **int**-Wert ohne Vorzeichenerweiterung und liefert diesen zurück. Bei Erreichen des Dateiendes oder im Fehlerfall wird der Wert EOF zurückgeliefert.

Siehe auch — *fgetchar, fputc, getc, getch, getchar, getche, ungetc, ungetch*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   FILE *stream;
   char string[] = "This is a test";
   char ch;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write a string into the file */
   fwrite(string, strlen(string), 1, stream);

   /* seek to the beginning of the file */
   fseek(stream, 0, SEEK_SET);

   do
   {
      /* read a char from the file */
      ch = fgetc(stream);

      /* display the character */
      putch(ch);
   } while (ch != EOF);
```

```
   fclose(stream);
   return 0;
}
```

fgetchar

Funktion Liest ein Zeichen aus *stdin*.

Syntax
```
#include <stdio.h>
int fgetchar(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *fgetchar* liest das jeweils nächste Zeichen aus *stdin* und liefert es zurück. Sie ist definiert als *fgetc*(*stdin*).

Rückgabewert *fgetchar* konvertiert bei fehlerfreier Ausführung das gelesene Zeichen in einen **int**-Wert ohne Vorzeichenerweiterung und liefert diesen zurück. Bei Erreichen des Dateiendes oder im Fehlerfall wird der Wert EOF zurückgeliefert.

Siehe auch *fgetc, fputchar, getchar*

Beispiel
```
#include <stdio.h>

int main(void)
{
   char ch;

   /* prompt the user for input */
   printf("Enter a character followed by <Enter>: ");

   /* read the character from stdin */
   ch = fgetchar();

   /* display what was read */
   printf("The character read is: '%c'\n", ch);
   return 0;
}
```

fgetpos

Funktion Ermittelt die aktuelle Position des Dateizeigers in einer Datei.

Syntax
```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *fgetpos* speichert die Position des zu *stream* gehörigen Dateizeigers in **pos*. Der exakte Wert des Dateizeigers ist für Ihre Zwecke irrelevant.

Der Datentyp *fpos_t* ist in stdio.h als **typedef long** *fpos_t;* definiert.

Rückgabewert *fgetpos* liefert bei fehlerfreier Ausführung 0 zurück.

Im Fall eines Fehlers liefert *fgetpos* einen Wert ungleich Null und setzt die globale Variable *errno* auf EBADF oder EINVAL.

Siehe auch *fseek, fsetpos, ftell, tell*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   FILE *stream;
   char string[] = "This is a test";
   fpos_t filepos;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write a string into the file */
   fwrite(string, strlen(string), 1, stream);

   /* report the file pointer position */
   fgetpos(stream, &filepos);
   printf("The file pointer is at byte %ld\n",
          filepos);

   fclose(stream);
   return 0;
}
```

fgets

Funktion Liest einen String aus einem Stream.

Syntax

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fgets* liest eine Zeichenfolge aus *stream* in den durch *s* angegebenen String und bricht ab, wenn entweder ein Zeilenvorschub (\n) gelesen wird oder n-1 Zeichen gelesen wurden. Der Zeilenvorschub wird am Ende von s gespeichert. Anschließend hängt *fgets* automatisch ein Nullzeichen (\0) an, um das Ende des Strings zu markieren.

Rückgabewert *fgets* liefert bei fehlerfreier Ausführung den durch *s* bezeichneten String zurück. Bei Erreichen des Dateiendes oder im Fehlerfall ist der Rückgabewert NULL.

Siehe auch *cgets, fputs, gets*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   FILE *stream;
   char string[] = "This is a test";
   char msg[20];

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write a string into the file */
   fwrite(string, strlen(string), 1, stream);

   /* seek to the start of the file */
   fseek(stream, 0, SEEK_SET);

   /* read a string from the file */
   fgets(msg, strlen(string)+1, stream);

   /* display the string */
   printf("%s", msg);

   fclose(stream);
   return 0;
}
```

filelength

Funktion Ermittelt die Länge einer Datei in Bytes.

Syntax

```
#include <io.h>
long filelength(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *filelength* liefert die Länge der über *handle* angegebenen Datei in Bytes zurück.

Rückgabewert Bei fehlerfreier Ausführung liefert *filelength* die Dateigröße in Bytes als **long**-Wert. Im Fall eines Fehlers liefert *filelength* -1 zurück und setzt die globale Variable *errno* auf

EBADF Ungültiges Handle

Siehe auch *fopen, lseek, open*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* create a file containing 10 bytes */
   handle = open("DUMMY.FIL", O_CREAT);
   write(handle, buf, strlen(buf));

   /* display the size of the file */
   printf("file length in bytes: %ld\n",
          filelength(handle));

   /* close the file */
   close(handle);
   return 0;
}
```

fileno

Funktion Liefert das Handle einer Datei zurück.

Syntax

```
#include <stdio.h>
int fileno(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *fileno* ist ein Makro und liefert das Handle des über *stream* angegebenen Stream zurück. Wenn *stream* mehr als ein Handle zugeordnet ist, wird das bei der ersten Öffnung vergebene Handle zurückgeliefert.

Rückgabewert *fileno* liefert das zu *stream* gehörende Handle als Integer zurück.

Siehe auch *fdopen, fopen, freopen*

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *stream;
   int handle;

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");
```

```
/* obtain the file handle associated with the stream */
   handle = fileno(stream);

   /* display the handle number */
   printf("handle number: %d\n", handle);

   /* close the file */
   fclose(stream);
   return 0;
}
```

fillellipse

Funktion — Zeichnet eine ausgefüllte Ellipse.

Syntax

```
#include <graphics.h>
void far fillellipse(int x, int y, int xradius,
                     int yradius);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *fillellipse* zeichnet eine Ellipse mit *x,y* als Mittelpunkt sowie *xradius* als horizontale und *yradius* als vertikale Achse und füllt sie mit dem aktuellen Füllmuster und der aktuellen Füllfarbe aus.

Rückgabewert — Keiner.

Siehe auch — *arc, circle, ellipse, pieslice,*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy, i;
  int xradius = 100, yradius = 50;

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n",
           grapherrormsg(errorcode));
    printf("Press any key to halt:");
```

```
    getch();
    exit(1);       /* terminate with an error code */
  }

  midx = getmaxx() / 2;
  midy = getmaxy() / 2;

  /* loop through the fill patterns */
  for (i = EMPTY_FILL; i < USER_FILL; i++) {
    /* set fill pattern */
    setfillstyle(i, getmaxcolor());

    /* draw a filled ellipse */
    fillellipse(midx, midy, xradius, yradius);
    getch();
  }

  /* clean up */
  closegraph();
  return 0;
}
```

fillpoly

Funktion: Zeichnet ein ausgefülltes Polygon.

Syntax:

```
#include <graphics.h>
void far fillpoly(int numpoints, int far *polypoints);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung: *fillpoly* zeichnet den Umriß eines Polygons mit *numpoints* Eckpunkten in der aktuellen Linienart und -farbe (genauso wie *drawpoly*) und füllt es dann mit dem aktuellen Füllmuster und der aktuellen Füllfarbe aus.

polypoints zeigt auf eine Folge von (*numpoints* * 2) Integerwerten. Jedes Integerpaar steht für *x*- und *y*-Koordinaten eines Polygonpunktes.

Rückgabewert: Keiner.

Siehe auch: *drawpoly, floodfill, graphresult, setfillstyle*

Beispiel:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
```

```
/* request auto detection */
int gdriver = DETECT, gmode, errorcode;
int i, maxx, maxy;

/* our polygon array */
int poly[8];

/* initialize graphics, local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk)
/* an error occurred */
{
   printf("Graphics error: %s\n",
          grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);
   /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();

poly[0] = 20;        /* 1st vertex */
poly[1] = maxy / 2;

poly[2] = maxx - 20; /* 2nd */
poly[3] = 20;

poly[4] = maxx - 50; /* 3rd */
poly[5] = maxy - 20;

/*
   4th vertex. fillpoly automatically
   closes the polygon.
*/
poly[6] = maxx / 2;
poly[7] = maxy / 2;

/* loop through the fill patterns */
for (i=EMPTY_FILL; i<USER_FILL; i++)
{
   /* set fill pattern */
   setfillstyle(i, getmaxcolor());
```

```
      /* draw a filled polygon */
      fillpoly(4, poly);

      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

findfirst

Funktion Sucht ein Dateiverzeichnis ab.

Syntax
```
#include <dir.h>
#include <dos.h>
int findfirst(const char *pathname,
              struct ffblk *ffblk, int attrib);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *findfirst* verwendet die DOS-Funktion *0x4E*, um ein Verzeichnis nach dem ersten Vorkommen eines Dateieintrags abzusuchen.

Der Parameter *pathname* ist ein String und bezeichnet den zu suchenden Eintrag. Er kann optional eine Laufwerksbezeichnung und/oder einen Suchweg enthalten, Jokerzeichen (* und ?) sind erlaubt. Falls eine Datei gefunden wird, werden in der über *ffblk* angegebene Struktur die Datei- und Verzeichnisinformationen gespeichert.

Die Struktur *ffblk* ist folgendermaßen definiert:

```
struct ffblk {
    char ff_reserved[21];  /* von DOS verwendet */
    char ff_attrib;        /* Attribut */
    int  ff_ftime;         /* Uhrzeit der Erstellung*/
    int  ff_fdate;         /* Datum der Erstellung */
    long ff_fsize;         /* Dateigröße in Bytes */
    char ff_name[13];      /* Name der Datei */
};
```

Über den Parameter *attrib* wird das Attribut des zu suchenden Eintrags angegeben. *attrib* kann eine der folgenden Konstanten sein, die in dos.h definiert sind:

FA_RDONLY	Nur-Lesen-Attribut
FA_HIDDEN	Verborgene Datei

FA_SYSTEM	Systemdatei
FA_LABEL	Datenträgername
FA_DIREC	Verzeichnis
FA_ARCH	Archiv-Bit

Ausführliche Informationen über diese Attribute finden Sie in Ihren DOS-Handbüchern.

Beachten Sie, daß *ff_ftime* und *ff_fdate* Bitfelder für die aktuelle Uhrzeit und das aktuelle Datum enthalten. Die Struktur dieser Bitfelder ist von DOS vorgegeben. Es handelt sich in beiden Fällen um 16 Bit große Strukturen, die in drei Felder unterteilt sind:

ff_time:

Bits 0 bis 4	Resultierende Sekunden geteilt durch 2 (d.h. 10 bedeutet hier 20 Sekunden)
Bits 5 bis 10	Minuten
Bits 11 bis 15	Stunden

ff_date:

Bits 0 bis 4	Tag
Bits 5 bis 8	Monat
Bits 9 bis 15	Jahre seit 1980 (d.h. 9 bedeutet hier 1989)

Die in io.h definierte Struktur *ftime* verwendet Datums- und Uhrzeit-Bitfelder ähnlicher Struktur wie *ff_fdate* und *ff_ftime* (vgl. *getftime* oder *setftime*).

Rückgabewert *findfirst* liefert den Wert 0 zurück, wenn ein zu *pathname* passender Eintrag gefunden wurde. Wenn kein Eintrag gefunden wird oder der angegebene Dateiname aus irgendeinem Grund ungültig ist, wird -1 als Ergebnis geliefert und *errno* bekommt den Wert:

ENOENT	Datei/Suchweg nicht gefunden

doserrno wird in diesem Fall auf einen der folgenden Werte gesetzt:

ENOENT	Datei/Suchweg nicht gefunden
ENMFILE	Keine weiteren passenden Einträge

Siehe auch *findnext*

Beispiel

```
/* findfirst and findnext example */

#include <stdio.h>
#include <dir.h>

int main(void)
{
   struct ffblk ffblk;
   int done;
```

```
   printf("Directory listing of *.*\n");
   done = findfirst("*.*",&ffblk,0);
   while (!done)
   {
      printf("  %s\n", ffblk.ff_name);
      done = findnext(&ffblk);
   }

   return 0;
}
```

Programm-ausgabe:

```
Directory listing of *.*
   FINDFRST.C
   FINDFRST.OBJ
   FINDFRST.EXE
```

findnext

Funktion: Setzt eine *findfirst*-Suche fort.

Syntax:

```
#include <dir.h>
int findnext(struct ffblk *ffblk);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung: Mit *findnext* kann ein Verzeichnis nach weiteren Einträgen abgesucht werden, die zu *pathname* einer vorangegangenen *findfirst*-Suche passen.

Als Argument muß die Struktur des Typs *ffblk* übergeben werden, die durch *findfirst* gesetzt wurde, da diese notwendige Informationen zur Fortsetzung der Suche enthält. *findnext* liefert bei jedem Aufruf einen weiteren Dateieintrag über *ffblk* zurück – solange, bis das Verzeichnis keine weiteren Einträge mehr enthält, die dem *pathname* des Aufrufs von *findfirst* entsprechen.

Rückgabewert: *findnext* liefert den Wert 0 zurück, wenn ein weiterer zu *pathname* passender Eintrag gefunden wurde. Wenn keine weiteren Einträge mehr vorhanden sind oder der angegebene Dateiname aus irgendeinem Grund ungültig ist, wird -1 geliefert und *errno* bekommt den Wert:

ENOENT	Datei/Suchweg nicht gefunden

doserrno wird auf einen der folgenden Werte gesetzt:

ENOENT	Datei/Suchweg nicht gefunden
ENMFILE	Keine weiteren passenden Einträge

Siehe auch: *findfirst*

Beispiel

```
/* findfirst and findnext example */

#include <stdio.h>
#include <dir.h>

int main(void)
{
   struct ffblk ffblk;
   int done;
   printf("Directory listing of *.*\n");
   done = findfirst("*.*",&ffblk,0);
   while (!done)
   {
      printf("  %s\n", ffblk.ff_name);
      done = findnext(&ffblk);
   }

   return 0;
}
```

Programm-ausgabe:

```
Directory listing of *.*
   FINDFRST.C
   FINDFRST.OBJ
   FINDFRST.EXE
```

floodfill

Funktion

Füllt einen umschlossenen Bereich mit dem aktuellen Füllmuster.

Syntax

```
#include <graphics.h>
void far floodfill(int x, int y, int border);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

floodfill arbeitet im Gegensatz zu *fillpoly* bit-orientiert; ausgehend vom Punkt (*x*,*y*) wird der Bildschirm in alle Richtungen mit der aktuellen Füllfarbe im aktuellen Muster gefüllt, wobei das Erreichen eines Pixels mit der Farbe *border* die Operation in der jeweiligen Richtung abbricht.

Wenn sich der Punkt (*x*,*y*) innerhalb einer Fläche befindet, die mit Linien der Farbe *border* umschlossen ist, füllt *floodfill* diese Fläche. Befindet sich (*x*,*y*) außerhalb dieser Fläche, dann wird der restliche Bildschirm (bzw. das restliche Zeichenfenster) gefüllt, und die umschlossene Fläche bleibt frei.

Um die Kompatibilität mit späteren Versionen des Compilers zu erhalten, sollte *fillpoly* anstelle von *floodfill* verwendet werden (soweit möglich).

Hinweis: Diese Funktion ist nicht zusammen mit dem IBM 8514-Treiber verwendbar.

Rückgabewert *floodfill* liefert keinen Wert zurück. Wenn während des Flächenfüllens ein Fehler auftritt, liefert ein nachfolgender Aufruf von *graphresult* den Wert -7.

Siehe auch *drawpoly, fillpoly, graphresult, setcolor, setfillstyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* initialize graphics, local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)
   /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
      /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();

   /* select drawing color */
   setcolor(getmaxcolor());

   /* select fill color */
   setfillstyle(SOLID_FILL, getmaxcolor());

   /* draw a border around the screen */
   rectangle(0, 0, maxx, maxy);

   /* draw some circles */
   circle(maxx / 3, maxy /2, 50);
   circle(maxx / 2, 20, 100);
   circle(maxx-20, maxy-50, 75);
   circle(20, maxy-20, 25);
```

```
   /* wait for a key */
   getch();

   /* fill in bounded region */
   floodfill(2, 2, getmaxcolor());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

floor, floorl

Funktion — Rundet ab.

Syntax

```
#include <math.h>
double floor(double x);
long double floorl(long double x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*floor*
x		x			*floorl*

Beschreibung — *floor* rundet auf den größten Integer-Wert ab, der nicht größer als *x* ist.

floorl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert — *floor* liefert den Integer als **double**-Wert zurück.

floorl liefert den Integer als Wert vom Typ **long double** zurück.

Siehe auch — *ceil, fmod*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double number = 123.54;
   double down, up;

   down = floor(number);
   up = ceil(number);

   printf("original number     %10.2lf\n", number);
   printf("number rounded down %    10.2lf\n", down);
```

```
   printf("number rounded up   %10.2lf\n", up);
   return 0;
}
```

flushall

Funktion Entleert die Puffer aller Streams.

Syntax
```
#include <stdio.h>
int flushall(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *flushall* löscht alle zu offenen Eingabe-Streams gehörigen Puffer und schreibt alle zu offenen Ausgabe-Streams gehörigen Puffer in ihre jeweiligen Dateien. Die auf *flushall* folgenden Leseoperationen lesen aus ihren Eingabedateien neue Daten in die Puffer ein.

Durch *flushall* werden keine Streams geschlossen.

Rückgabewert *flushall* liefert die Gesamtzahl der offenen Ein- und Ausgabedateien als Integerwert zurück.

Siehe auch *fclose, fcloseall, fflush*

Beispiel
```
#include <stdio.h>

int main(void)
{
   FILE *stream;

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");

   /* flush all open streams */
   printf("%d streams were flushed.\n", flushall());

   /* close the file */
   fclose(stream);
   return 0;
}
```

_fmem...

Siehe mem...

fmod, fmodl

Funktion Berechnet *x* Modulo *y*, den Restwert der Division *x/y*.

Syntax
```
#include <math.h>
double fmod(double x, double y);
long double fmodl(long double x, long double y);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*fmod*
x		x			*fmodl*

Beschreibung *fmod* berechnet *x* Modulo *y* (den Rest *f*, wobei $x = ay + f$ für einen Integer *a* und $0 <= f < y$ gilt).

fmodl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert *fmod* und *fmodl* liefern den Rest *f*, wobei $x = ay + f$ gilt (wie beschrieben). Wenn $y = 0$ ist, liefern *fmod* und *fmodl* den Wert 0 zurück.

Siehe auch *ceil, floor, modf*

Beispiel
```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double x = 5.0, y = 2.0;
   double result;

   result = fmod(x,y);
   printf("The remainder of (%lf / %lf) is %lf\n",
          x, y, result);
   return 0;
}
```

Programmausgabe:
```
The remainder of 5.0 / 2.0 is 1.0
```

fnmerge

Funktion Erzeugt einen Pfadnamen aus einzelnen Komponenten.

Syntax
```
#include <dir.h>
void fnmerge(char *path, const char *drive,
             const char *dir, const char *name,
             const char *ext);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *fnmerge* erzeugt einen Pfadnamen aus Einzelkomponenten und speichert ihn in dem durch *path* bezeichneten String.

Der neue Pfadname heißt

```
X:\DIR\SUBDIR\NAME.EXT
```

wobei gilt:

drive = X
dir = \DIR\SUBDIR\
name = NAME
ext = .EXT

fnmerge setzt voraus, daß der mit *path* übergebene String eine ausreichende Größe für den konstruierten Pfadnamen hat. Die Maximalgröße ist MAXPATH, wobei MAXPATH in dir.h definiert ist.

fnmerge und *fnsplit* verhalten sich invers. Die Anwendung von *fnmerge* auf die durch *fnsplit* ermittelten Elemente eines vollständigen Pfadnamens ergeben wieder das Original.

Rückgabewert Keiner.

Siehe auch *fnsplit*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <dir.h>

int main(void)
{
   char s[MAXPATH];
   char drive[MAXDRIVE];
   char dir[MAXDIR];
   char file[MAXFILE];
   char ext[MAXEXT];

   getcwd(s,MAXPATH);
          /* get the current working directory */
   strcat(s,"\\");
          /* append on a trailing character */
   fnsplit(s,drive,dir,file,ext);
          /* split the string to separate elems */
   strcpy(file,"DATA");
   strcpy(ext,".TXT");
   fnmerge(s,drive,dir,file,ext);
          /* merge everything into one string */
   puts(s);
```

```
            /* display resulting string */

   return 0;
}
```

fnsplit

Funktion — Zerlegt einen vollständigen Pfadnamen in Einzelkomponenten.

Syntax

```
#include <dir.h>
int fnsplit(const char *path, char *drive, char *dir,
            char *name, char *ext);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *fnsplit* übernimmt einen vollständigen Dateinamen (Parameter *path*) als String in der Form

```
X:\DIR\SUBDIR\NAME.EXT
```

und zerlegt *path* in seine vier Einzelkomponenten. Diese werden dann in den durch *drive*, *dir*, *name* und *ext* bezeichneten Strings gespeichert. (Alle fünf Parameter müssen übergeben werden, einzelne davon dürfen aber NULL sein, was zur Folge hat, daß die zugehörige Komponente zwar analysiert, aber nicht gespeichert wird.)

Die Maximalgrößen aller beteiligen Strings sind in dir.h als Konstanten mit folgenden Werten definiert, wobei jede Größe Platz für das nullterminierende Zeichen einschließt:

Konstante	max.	String
MAXPATH	80	*path*
MAXDRIVE	3	*drive* inklusive Doppelpunkt (:)
MAXDIR	66	*dir* inklusive führendem und abschließendem umgekehrten Schrägstrich (\)
MAXFILE	9	*name*
MAXEXT	5	*ext* inklusive führendem Punkt (.)

fnsplit unterstellt, daß die durch *drive*, *dir*, *name* und *ext* bezeichneten Strings eine ausreichende Größe haben, um jede Komponente zu speichern, die nicht NULL ist.

fnsplit verteilt die Interpunktionszeichen aus *path* in folgender Weise:

drive enthält den Laufwerksbezeichner mit einem Doppelpunkt (C:, A:, usw.);

dir enthält den führenden und den abschließenden umgekehrten Schrägstrich (\BC\include\, \source\, usw.);

name enthält den Dateinamen;

ext enthält den Punkt, der die Namenserweiterung einleitet (.C, .EXE, usw.).

fnmerge und *fnsplit* verhalten sich invers. Die Anwendung von *fnmerge* auf die durch *fnsplit* ermittelten Elemente eines vollständigen Pfadnamens ergeben wieder das Original.

Rückgabewert *fnsplit* liefert einen Integer zurück, der sich aus fünf in dir.h definierten Flags zusammensetzt. Ein gesetztes Flag zeigt an, daß die zugehörige Komponente in *path* vorhanden war. Folgende Flags sind in dir.h definiert:

EXTENSION	Eine Namenserweiterung
FILENAME	Ein Dateiname
DIRECTORY	Mindestens ein Verzeichnis
DRIVE	Ein Laufwerksbezeichner
WILDCARDS	Jokerzeichen (* und/oder ?)

Siehe auch *fnmerge*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>
#include <dir.h>

int main(void)
{
    char *s;
    char drive[MAXDRIVE];
    char dir[MAXDIR];
    char file[MAXFILE];
    char ext[MAXEXT];
    int flags;

    s=getenv("COMSPEC");
         /* get the comspec environment parameter */
    flags=fnsplit(s,drive,dir,file,ext);

    printf("Command processor info:\n");
    if(flags & DRIVE)
       printf("\tdrive: %s\n",drive);
    if(flags & DIRECTORY)
       printf("\tdirectory: %s\n",dir);
    if(flags & FILENAME)
       printf("\tfile: %s\n",file);
    if(flags & EXTENSION)
       printf("\textension: %s\n",ext);

    return 0;
}
```

fopen

Funktion Öffnet einen Stream.

Syntax

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fopen* öffnet die durch *filename* bezeichnete Datei und ordnet ihr einen Stream zu. Zurückgeliefert wird ein Zeiger auf diesen Stream, über den weitere Operationen mit dieser Datei ausgeführt werden können.

Der Parameter *mode* kann einen der folgenden Werte annehmen:

r Öffnung ausschließlich für Leseoperationen.

w Erzeugung einer Datei für Schreiboperationen. Falls die Datei bereits existiert, wird sie überschrieben.

a Öffnung für Schreiboperationen. Falls die Datei bereits existiert, werden zusätzliche Informationen an das aktuelle Dateiende angefügt, ansonsten wird sie automatisch erzeugt.

r+ Öffnung einer Datei für Lese- und Schreiboperationen. Die Datei muß bereits existieren.

w+ Erzeugung einer neuen Datei für Lese- und Schreiboperationen. Falls die Datei bereits existiert, wird sie überschrieben.

a+ Öffnung einer Datei zum Lesen und »Anhängen« neuer Daten durch Schreiboperationen. Falls die Datei bereits existiert, bleibt ihr alter Inhalt erhalten – falls nicht, wird sie automatisch erzeugt.

Bei der Öffnung kann zusätzlich explizit angegeben werden, ob die Datei im Modus *text* (t) oder *binary* (b) bearbeitet werden soll. Die Angabe geschieht durch ein einfaches Anhängen des jeweiligen Buchstabens (also *wb*, *w+b*, *rt*, *a+t* etc.). Im Parameter *mode* kann bei *fopen* das Zeichen *t* bzw. *b* auch zwischen den Buchstaben und dem Pluszeichen eingefügt werden. Beispielsweise ist *rt+* identisch mit *r+t*.

Wenn die Angabe *b* oder *t* im *mode*-String fehlt, wird der jeweilige Modus durch die globale Variable *_fmode* bestimmt, deren Wert über die direkte Zuweisung einer der in fcntl.h definierten Konstanten O_TEXT oder O_BINARY festgelegt werden kann.

Mit *r+* und *w+* eröffnete Dateien können sowohl gelesen als auch beschrieben werden – zwischen einem Wechsel von »Schreiben« nach »Lesen« muß allerdings ein Aufruf von *fseek*

oder *rewind* stattfinden. Zwischen einem Wechsel von »Lesen« nach »Schreiben« muß ebenfalls *fseek* oder *rewind* stattfinden, es sei denn, die Leseoperation hat das Dateiende erreicht.

Rückgabewert *fopen* liefert bei fehlerfreier Ausführung einen Zeiger auf den neu geöffneten Stream, im Falle eines Fehlers den Wert NULL zurück.

Siehe auch *creat*, *dup*, *fclose*, *fdopen*, *ferror*, *_fmode* (globale Variable), *fread*, *freopen*, *fseek*, *fwrite*, *open*, *rewind*, *setbuf*, *setmode*

Beispiel

```
/* Program to create backup of the AUTOEXEC.BAT file */

#include <stdio.h>

int main(void)
{
   FILE *in, *out;

   if ((in = fopen("\\AUTOEXEC.BAT", "rt"))
       == NULL)
   {
      fprintf(stderr, "Cannot open input file.\n");
      return 1;
   }

   if ((out = fopen("\\AUTOEXEC.BAK", "wt"))
       == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }

   while (!feof(in))
      fputc(fgetc(in), out);

   fclose(in);
   fclose(out);
   return 0;
}
```

FP_OFF, FP_SEG

Funktion Ermittelt den Offset- oder Segment-Anteil einer **far**-Adresse.

Syntax

```
#include <dos.h>
unsigned FP_OFF(void far *p);
unsigned FP_SEG(void far *p);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *FP_OFF* ist ein Makro, das den Offset-Anteil der Adresse des **far**-Zeigers **p* ermittelt oder setzt.

FP_SEG ist ein Makro, das den Segment-Anteil des **far**-Zeigers **p* ermittelt oder setzt.

Rückgabewert *FP_OFF* liefert den Offset-Anteil eines Zeigers als **unsigned int** zurück.

FP_SEG liefert den Segment-Anteil eines Zeigers als **unsigned int** zurück.

Siehe auch *MK_FP*, *movedata*, *segread*

Beispiel

```
#include <stdio.h>
#include <dos.h>

main()
{
  char *str = "Hello\n";

  printf("The address pointed to by str is "
         "%04X:%04X\n", FP_SEG(str), FP_OFF(str));
  printf("The address of str is %04X:%04X\n",
         FP_SEG(&str), FP_OFF(&str));
  return 0;
}
```

_fpreset

Funktion Initialisiert die Gleitkommafunktionen neu.

Syntax

```
#include <float.h>
void _fpreset(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_fpreset* reinitialisiert die Gleitkommafunktionen von Borland C++ und wird normalerweise zusammen mit *system*, *exec...* oder *spawn...* verwendet.

Hinweis: Unter DOS kann ein Prozeß, der von *system*, *exec...* oder *spawn...* als untergeordneter Prozeß gestartet wird, den Gleitkommastatus seines übergeordneten Prozesses verändern. Wenn ein Koprozessor (80x87) verwendet wird, müssen die folgenden Vorsichtsmaßnahmen getroffen werden:

- Die Funktionen *system*, *exec...* und *spawn...* dürfen nicht aufgerufen werden, während ein Gleitkommaausdruck ausgewertet wird.

- Nach Aufrufen von *system*, *exec...* und *spawn...* sollten Koprozessor und Gleitkommafunktionen mit *_fpreset* zurück-

gesetzt werden, wenn die Möglichkeit besteht, daß der untergeordnete Prozeß Gleitkommaoperationen mit dem 80x87 durchgeführt hat.

Rückgabewert Keiner.

Siehe auch *_clear87, _control87, exec..., spawn..., _status87, system*

Beispiel

```
#include <stdio.h>
#include <float.h>
#include <setjmp.h>
#include <signal.h>
#include <process.h>
#include <conio.h>

#ifdef __cplusplus
    typedef void (*fptr)(int);
#else
    typedef void (*fptr)();
#endif

jmp_buf reenter;

/* define a handler for trapping floating
   point errors */
void float_trap(int sig)
{
   printf("Trapping floating point error,");
   printf("signal is %d\n",sig);
   printf("Press a key to continue\n");
   getch();
/*
   reset the 8087 chip or emulator to clear
   any extraneous garbage
*/
   _fpreset();
/* return to the problem spot */
   longjmp(reenter, -1);
}

int main(void)
{
   float one = 3.14, two = 0.0;

/*
   install signal handler for floating point
   exception
*/
   signal(SIGFPE, (fptr)float_trap);
   printf("Generating a math error,");
   printf("press a key\n");
   getch();
```

```
   if (setjmp(reenter) == 0)
      one /= two;
   printf("Returned from signal trap\n");
   return 0;
}
```

fprintf

Funktion

Schreibt formatiert in einen Stream.

Syntax

```
#include <stdio.h>
int fprintf(FILE *stream,
            const char *format[,argument,...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung

fprintf übernimmt eine Folge von Argumenten, wendet auf jedes den im Formatstring (Parameter *format*) festgelegten Formatbezeichner an und gibt die formatierten Daten zu einem Stream aus. Die Anzahl der Formatbezeichner und der Argumente muß gleich sein.

Hinweis: Ausführliche Informationen zu Formatbezeichnern finden Sie in der Beschreibung von *printf*.

Rückgabewert

fprintf liefert bei fehlerfreier Ausführung die Anzahl der ausgegebenen Bytes, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch

cprintf, fscanf, printf, putc, sprintf

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *stream;
   int i = 100;
   char c = 'C';
   float f = 1.234;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write some data to the file */
   fprintf(stream, "%d %c %f", i, c, f);

   /* close the file */
   fclose(stream);
   return 0;
}
```

fputc

Funktion Gibt ein Zeichen zu einem Stream aus.

Syntax
```
#include <stdio.h>
int fputc(int c, FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fputc* gibt das Zeichen *c* zu dem durch *stream* angegebenen Stream aus.

Rückgabewert *fputc* liefert das übergebene Zeichen *c*, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch *fgetc, putc*

Beispiel
```
#include <stdio.h>

int main(void)
{
   char msg[] = "Hello world";
   int i = 0;

   while (msg[i])
   {
      fputc(msg[i], stdout);
      i++;
   }
   return 0;
}
```

fputchar

Funktion Gibt ein Zeichen zu *stdout* aus.

Syntax
```
#include <stdio.h>
int fputchar(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *fputchar* gibt das Zeichen *c* zu *stdout* aus und hat dieselbe Wirkung wie *fputc* (*c, stdout*).

Rückgabewert *fputchar* liefert bei fehlerfreier Ausführung das übergebene Zeichen *c*, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch *fgetchar, putchar*

Beispiel

```
#include <stdio.h>

int main(void)
{
   char msg[] = "This is a test";
   int i = 0;

   while (msg[i])
   {
      fputchar(msg[i]);
      i++;
   }
   return 0;
}
```

fputs

Funktion Gibt einen String in einen Stream aus.

Syntax

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fputs* schreibt den nullterminierten String *s* in den angegebenen Ausgabe-Stream.

Diese Funktion erzeugt *keinen* automatischen Zeilenvorschub am Ende der Ausgabe und das abschließende Nullzeichen wird nicht ausgegeben.

Rückgabewert Bei fehlerfreier Ausführung liefert *fputs* das zuletzt geschriebene Zeichen (einen positiven Wert), im Falle eines Fehlers den Wert EOF zurück.

Siehe auch *fgets, gets, puts*

Beispiel

```
#include <stdio.h>

int main(void)
{
   /* write a string to standard output */
   fputs("Hello world\n", stdout);

   return 0;
}
```

fread

Funktion — Liest Daten aus einem Stream.

Syntax

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t n,
             FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *fread* liest *n* Datenelemente, die jeweils die Größe *size* Bytes haben, aus dem angegebenen Stream in den Speicherbereich, auf den *ptr* zeigt. Es werden insgesamt (*n* * *size*) Bytes gelesen.

Rückgabewert — *fread* liefert bei fehlerfreier Ausführung die Anzahl der gelesenen Datenelemente (nicht Bytes) zurück. Im Falle eines Fehlers oder am Dateiende wird die Anzahl von Elementen (möglicherweise 0) zurückgeliefert, die noch fehlerfrei gelesen werden konnte.

Siehe auch — *fopen, fwrite, printf, read*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   FILE *stream;
   char msg[] = "this is a test";
   char buf[20];

   if ((stream = fopen("DUMMY.FIL", "w+"))
       == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }

   /* write some data to the file */
   fwrite(msg, strlen(msg)+1, 1, stream);

   /* seek to the beginning of the file */
   fseek(stream, SEEK_SET, 0);

   /* read the data and display it */
   fread(buf, strlen(msg)+1, 1, stream);
   printf("%s\n", buf);

   fclose(stream);
   return 0;
}
```

free

Funktion Gibt einen belegten Speicherblock wieder frei.

Syntax
```
#include <alloc.h>
void free(void *block);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *free* gibt einen zuvor mit *calloc, malloc* oder *realloc* belegten Block wieder frei.

Rückgabewert Keiner.

Siehe auch *calloc, freemem, malloc, realloc, strdup*

Beispiel
```
#include <string.h>
#include <stdio.h>
#include <alloc.h>

int main(void)
{
   char *str;

   /* allocate memory for string */
   str = (char *) malloc(10);

   /* copy "Hello" to string */
   strcpy(str, "Hello");

   /* display string */
   printf("String is %s\n", str);

   /* free memory */
   free(str);

   return 0;
}
```

freemem, _dos_freemem

Funktion Gibt einen vorher belegten DOS-Speicherblock wieder frei.

Syntax
```
#include <dos.h>
int freemem(unsigned segx);
unsigned _dos_freemem(unsigned segx);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *freemem* gibt einen zuvor mit *allocmem* belegten Speicherblock wieder frei. *_dos_freemem* gibt einen zuvor mit *_dos_allocmem*

belegten Speicherblock wieder frei. Die Segmentadresse des Blocks wird über den Parameter *segx* angegeben.

Rückgabewert *freemem* und *_dos_freemem* liefern bei fehlerfreier Ausführung den Wert 0 zurückt.

Im Fall eines Fehlers liefert *freemem* den Wert -1 und *_dos_freemem* den DOS-Fehlercode zurück. Die globale Variable *errno* erhält folgenden Wert:

ENOMEM Nicht genug Platz im Hauptspeicher

Siehe auch *allocmem, _dos_allocmem, free*

Beispiel

```
#include <dos.h>
#include <alloc.h>
#include <stdio.h>

int main(void)
{
   unsigned int size, segp;
   int stat;

   size = 64; /* (64 x 16) = 1024 bytes */
   stat = allocmem(size, &segp);
   if (stat < 0)
      printf("Allocated memory at segment: %x\n",
             segp);
   else
      printf("Failed: maximum number of\
             paragraphs available is %u\n", stat);
   freemem(segp);

   return 0;
}
```

Beispiel 2

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
  unsigned int size, segp, err, maxb;
  size = 64; /* (64 x 16) = 1024 bytes */
  err = _dos_allocmem(size, &segp);
  if (err == 0)
    printf("Allocated memory at segment: %x\n", segp);
  else {
    perror("Unable to allocate block");
    printf("Maximum no. of paragraphs available "
           "is %u\n", segp);
    return 1;
  }
  if (_dos_setblock(size * 2, segp, &maxb) == 0)
    printf("Expanded memory block at segment: %X\n",
```

```
            segp);
  else {
    perror("Unable to expand block");
    printf("Maximum no. of paragraphs available "
           "is %u\n", maxb);
  }
  _dos_freemem(segp);
  return 0;
}
```

freopen

Funktion Ordnet einem offenen Stream eine neue Datei zu.

Syntax

```
#include <stdio.h>
FILE *freopen(const char *filename, const char *mode,
              FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *freopen* ersetzt *stream* durch die angegebene Datei. Das Schließen des Stream findet unabhängig vom Erfolg des anschließenden Öffnens statt. In den meisten Fällen wird diese Funktion verwendet, um *stdin*, *stdout* und *stderr* eine andere Datei zuzuordnen.

Der Parameterstring *mode* kann einen der folgenden Werte annehmen:

r Öffnung ausschließlich für Leseoperationen.

w Erzeugung einer Datei für Schreiboperationen.

a Öffnung für Schreiboperationen. Falls die Datei bereits existiert, werden zusätzliche Informationen an das aktuelle Dateiende angefügt, ansonsten wird sie automatisch erzeugt.

r+ Öffnung einer Datei für Lese- und Schreiboperationen. Die Datei muß bereits existieren.

w+ Erzeugung einer neuen Datei für Lese- und Schreiboperationen.

a+ Öffnung einer Datei zum Lesen und »Anhängen« neuer Daten durch Schreiboperationen. Falls die Datei bereits existiert, bleibt ihr alter Inhalt erhalten – falls nicht, wird sie automatisch erzeugt.

Bei der Öffnung kann zusätzlich explizit angegeben werden, ob die Datei im Modus *text* (t) oder *binary* (b) bearbeitet werden soll. Die Angabe geschieht durch ein einfaches Anhän-

gen des jeweiligen Buchstabens (also *wb*, *w+b*, *rt*, *a+t* etc.). Wenn sie fehlt, wird der jeweilige Modus durch die globale Variable *_fmode* bestimmt, deren Wert über die direkte Zuweisung einer der in fcntl.h definierten Konstanten O_TEXT oder O_BINARY festgelegt werden kann.

Mit *r+* und *w+* eröffnete Dateien können sowohl gelesen als auch beschrieben werden – zwischen einem Wechsel von »Schreiben« nach »Lesen« muß allerdings ein *fseek* oder *rewind* stattfinden. Zwischen einem Wechsel von »Lesen« nach »Schreiben« muß ebenfalls *fseek* oder *rewind* stattfinden, es sei denn, die Leseoperation hat das Dateiende erreicht.

Rückgabewert *freopen* liefert bei fehlerfreier Ausführung das Argument *stream*, im Fall eines Fehlers den Wert NULL zurück.

Siehe auch *fclose*, *fdopen*, *fopen*, *open*, *setmode*

Beispiel

```
#include <stdio.h>

int main(void)
{
   /* redirect standard output to a file */
   if (freopen("OUTPUT.FIL", "w", stdout)
       == NULL)
      fprintf(stderr, "error redirecting stdout\n");

   /* this output will go to a file */
   printf("This will go into a file.");

   /* close the standard output stream */
   fclose(stdout);

   return 0;
}
```

frexp, frexpl

Funktion Ermittelt die Mantisse und den Exponenten einer Gleitkommazahl.

Syntax

```
#include <math.h>
double frexp(double x, int *exponent);
long double frexpl(long double x, int *exponent);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*frexp*
x		x			*frexpl*

Beschreibung *frexp* berechnet zu x die Mantisse m (ein **double**-Wert, dessen absoluter Betrag größer/gleich 0.5 und kleiner als 1.0 ist) und den Integerwert n so, daß die Gleichung $x = m * 2^n$ gilt. Der

Exponent *n* wird in dem Integer, auf den der Parameter *exponent* zeigt, gespeichert.

frexpl ist die **long double** Version. Sie erwartet für *x* ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert *frexp* und *frexpl* liefern die Mantisse *m* zurück. Die Behandlung von Fehlern kann für diese Funktionen über die Routinen *matherr* bzw. *_matherrl* modifiziert werden.

Siehe auch *exp*, *ldexp*

Beispiel

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double mantissa, number;
   int exponent;

   number = 8.0;
   mantissa = frexp(number, &exponent);

   printf("The number %lf is ", number);
   printf("%lf times two to the ", mantissa);
   printf("power of %d\n", exponent);

   return 0;
}
```

fscanf

Funktion Liest und formatiert Eingabe aus einem Stream.

Syntax

```
#include <stdio.h>
int fscanf(FILE *stream,
           const char *format[,adress,...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fscanf* liest eine Reihe von Eingabefeldern zeichenweise aus einem Stream. Entsprechend den Formatanweisungen, die *fscanf* im Formatstring *format* übergeben wurden, wird jedes Feld gelesen, formatiert und an der Adresse gespeichert, die nach *format* übergeben wurde. Die Anzahl der Formatanweisungen, der übergebenen Adressen und der Eingabefelder müssen übereinstimmen.

Hinweis: Unter der Funktion *scanf* finden Sie eine ausführliche Beschreibung der Formatanweisungen.

fscanf kann das Lesen eines bestimmten Feldes beenden,

bevor das normale Feldende (Whitespace) erreicht ist oder und aus verschiedenen Gründen die Leseoperation vollständig beenden. Mögliche Gründe werden bei *scanf* besprochen.

Rückgabewert *fscanf* liefert die Anzahl der fehlerfrei gelesenen, konvertierten und gespeicherten Felder zurück, wobei mit * unterdrückte Felder nicht gezählt werden.

Der Versuch, über das Ende einer Datei hinauszulesen, ergibt den Wert EOF. Wenn überhaupt keine Zuweisungen erfolgt sind, wird 0 zurückgeliefert.

Siehe auch *atof*, *cscanf*, *fprintf*, *printf*, *scanf*, *sscanf*, *vfscanf*, *vscanf*, *vsscanf*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   int i;

   printf("Input an integer: ");

   /* read an integer from the
      standard input stream */
   if (fscanf(stdin, "%d", &i))
      printf("The integer read was: %i\n", i);
   else
   {
      fprintf(stderr, "Error reading an integer "
              from stdin.\n");
      exit(1);
   }
   return 0;
}
```

fseek

Funktion Setzt den Dateizeiger eines Streams auf eine neue Position.

Syntax

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fseek* setzt den zu *stream* gehörigen Dateizeiger auf die neue Position, die *offset* Bytes von der durch *whence* gegebenen Dateiposition entfernt ist. Für Streams im Textmodus sollte *offset* 0 sein oder ein Wert, der von *ftell* geliefert wurde.

Die Art der Positionierung wird über drei in stdio.h definierte Konstanten für *whence* festgelegt:

SEEK_SET (Wert 0)	Relativ zum Dateianfang
SEEK_CUR (Wert 1)	Relativ zur vorherigen Position
SEEK_END (Wert 2)	Relativ zum Dateiende

Mit *ungetc* zurückgestellte Zeichen sind nach *fseek* nicht mehr erreichbar.

Benutzen Sie *fseek* für Stream-Ein-/Ausgabe und *lseek* für Datei-Handle-Ein-/Ausgabe.

In Dateien, die für Lese- *und* Schreiboperationen eröffnet worden sind, kann nach *fseek* zwischen »Lesen« und »Schreiben« gewechselt werden.

Rückgabewert *fseek* liefert bei fehlerfreier Ausführung den Wert 0 zurück, ansonsten einen Wert ungleich Null.

Hinweis: Es ist möglich, daß *fseek* 0 zurückliefert, obwohl bei der Ausführung der Funktion ein Fehler aufgetreten ist. Dies liegt daran, daß DOS, welches den Zeiger zurücksetzt, das Setzen nicht verifiziert. *fseek* gibt einen Fehlercode nur für ungeöffnete Dateien oder Geräte zurück.

Siehe auch *fgetpos*, *fopen*, *fsetpos*, *ftell*, *lseek*, *rewind*, *setbuf*, *tell*

Beispiel

```
#include <stdio.h>

long filesize(FILE *stream);

int main(void)
{
   FILE *stream;

   stream = fopen("MYFILE.TXT", "w+");
   fprintf(stream, "This is a test");
   printf("Filesize of MYFILE.TXT is %ld bytes\n",
          filesize(stream));
   fclose(stream);
   return 0;
}

long filesize(FILE *stream)
{
   long curpos, length;

   curpos = ftell(stream);
   fseek(stream, 0L, SEEK_END);
   length = ftell(stream);
   fseek(stream, curpos, SEEK_SET);
   return length;
}
```

fsetpos

Funktion — Positioniert den Dateizeiger eines Stream.

Syntax

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung — *fsetpos* setzt den Dateizeiger von *stream* auf eine neue Position. Die neue Position ist der Wert, der durch einen vorherigen Aufruf von *fgetpos* für diesen Stream ermittelt wurde. Das dazugehörige Dateiende-Flag wird gelöscht, zuvor mit *ungetc* »zurückgestellte« Zeichen sind nach dieser Operation nicht mehr verfügbar.

Die nächste Operation nach einem Aufruf von *fsetpos* kann sowohl eine Lese- als auch eine Schreiboperation sein.

Rückgabewert — *fsetpos* liefert bei fehlerfreier Ausführung den Wert 0 zurück. Im Fall eines Fehlers resultiert ein Wert ungleich Null und die globale Variable *errno* wird auf einen Wert ungleich Null gesetzt.

Siehe auch — *fgetpos*, *fseek*, *ftell*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

void showpos(FILE *stream);

int main(void)
{
   FILE *stream;
   fpos_t filepos;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* save the file pointer position */
   fgetpos(stream, &filepos);

   /* write some data to the file */
   fprintf(stream, "This is a test");

   /* show the current file position */
   showpos(stream);

   /* set a new file position, display it */
   if (fsetpos(stream, &filepos) == 0)
     showpos(stream);
   else
   {
```

```
      fprintf(stderr,"Error setting file pointer.\n");
      exit(1);
   }

   /* close the file */
   fclose(stream);
   return 0;
}

void showpos(FILE *stream)
{
   fpos_t pos;

   /* display the current file pointer
      position of a stream */
   fgetpos(stream, &pos);
   printf("File position: %ld\n", pos);
}
```

_fsopen

Funktion Öffnet einen Stream für den Modus SHARED.

Syntax

```
#include <stdio.h>
#include <share.h>
FILE *_fsopen(const char *filename, const char *mode,
              int shflg);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_fsopen* öffnet die durch *filename* angegebene Datei und ordnet ihr einen Stream zu. *_fsopen* liefert einen Zeiger zurück, mit dem der Stream bei anschließenden Operationen identifiziert werden kann.

Der Parameterstring *mode* kann einen der folgenden Werte annehmen:

r Öffnung ausschließlich für Leseoperationen.

w Erzeugung einer Datei für Schreiboperationen.

a Öffnung für Schreiboperationen. Falls die Datei bereits existiert, werden zusätzliche Informationen an das aktuelle Dateiende angefügt, ansonsten wird sie automatisch erzeugt.

r+ Öffnung einer Datei für Lese- und Schreiboperationen. Die Datei muß bereits existieren.

w+ Erzeugung einer neuen Datei für Lese- und Schreiboperationen. Existiert bereits eine Datei des angegebenen Namens, wird sie überschrieben.

a+ Öffnung einer Datei zum Lesen und »Anhängen« neuer Daten durch Schreiboperationen. Falls die Datei bereits existiert, bleibt ihr alter Inhalt erhalten – falls nicht, wird sie automatisch erzeugt.

Bei der Öffnung kann zusätzlich explizit angegeben werden, ob die Datei im Modus *text* (t) oder *binary* (b) bearbeitet werden soll. Die Angabe geschieht durch ein einfaches Anhängen des jeweiligen Buchstabens (also *wb*, *w+b*, *rt*, *a+t* etc.). Wenn sie fehlt, wird der jeweilige Modus durch die globale Variable *_fmode* bestimmt, deren Wert über die direkte Zuweisung einer der in fcntl.h definierten Konstanten O_TEXT oder O_BINARY festgelegt werden kann.

Mit *r+* und *w+* eröffnete Dateien können sowohl gelesen als auch beschrieben werden – zwischen einem Wechsel von »Schreiben« nach »Lesen« muß allerdings ein *fseek* oder *rewind* stattfinden. Zwischen einem Wechsel von »Lesen« nach »Schreiben« muß ebenfalls *fseek* oder *rewind* stattfinden, es sei denn, die Leseoperation hat das Dateiende erreicht.

shflag legt den Typ des gemeinsamen Zugriffs fest, der auf die Datei *filename* angewendet werden soll. Der DOS-Befehl SHARE muß ausgeführt worden sein, damit *shflag* nicht ignoriert wird. Die symbolischen Konstanten für *shflag* sind in share.h definiert:

SH_COMPAT	Setzt Kompatibilitätsmodus
SH_DENYRW	Lesen und Schreiben nicht erlaubt
SH_DENYWR	Schreiben nicht erlaubt
SH_DENYRD	Lesen nicht erlaubt
SH_DENYNONE	Lesen und Schreiben erlaubt
SH_DENYNO	Lesen und Schreiben erlaubt

Rückgabewert *_fsopen* liefert bei fehlerfreier Ausführung einen Zeiger auf den neu geöffneten Stream, im Fall eines Fehlers den Wert 0 zurück.

Siehe auch *creat*, *_dos_open*, *dup*, *fclose*, *fdopen*, *ferror*, *_fmode* (globale Variable), *fopen*, *fread*, *freopen*, *fseek*, *fwrite*, *open*, *rewind*, *setbuf*, *setmode*, *sopen*

Beispiel

```
#include <io.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
  FILE *f;
  int status;
  f = _fsopen("c:\\autoexec.bat", "r", SH_DENYNO);
  if (f == NULL)
```

```
  {
    printf("_fsopen failed\n");
    exit(1);
  }
  status = access("c:\\autoexec.bat", 6);
  if (status == 0)
    printf("read/write access allowed\n");
  else
    printf("read/write access not allowed\n");
  fclose(f);
  return 0;
}
```

fstat, stat

Funktion Liefert Informationen über eine offene Datei.

Syntax
```
#include <sys\stat.h>
int fstat(int handle, struct stat *statbuf);
int stat(char *path, struct stat *statbuf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *fstat* speichert in der Struktur *stat* Informationen über die offene Datei oder das Verzeichnis, die bzw. das mit *handle* angegeben ist.

stat speichert Informationen über die angegebene Datei oder das Verzeichnis in der Struktur *stat*.

statbuf zeigt auf die Struktur *stat*, die in sys\stat.h definiert ist und folgende Felder enthält:

st_mode	Bitmuster mit Informationen über den Modus der offenen Datei.
st_dev	Nummer des Laufwerks, das die Datei enthält oder ein Datei-Handle.
st_rdev	Wie *st_dev*.
st_nlink	immer eine Integerkonstante mit dem Wert 1.
st_size	Größe der offenen Datei in Bytes.
st_atime	Datum/Uhrzeit der letzten Änderung.
st_mtime	wie *st_atime*.
st_ctime	wie *st_atime*.

Die Struktur *stat* enthält noch drei weitere Felder, die hier nicht aufgeführt sind, da die darin enthaltenen Werte unter DOS nicht von Bedeutung sind.

Das Bitmuster *st_mode* setzt sich aus mehreren Einzelinformationen zusammen, die in sys\stat.h als Bit-Konstanten definiert sind.

Eines dieser beiden Bits ist gesetzt:

S_IFCHR Wenn *handle* sich auf ein Gerät bezieht;
S_IFREG Wenn es sich um eine reguläre Datei handelt.

Eines oder beide der folgenden Bits sind gesetzt:

S_IWRITE Wenn der Benutzer Schreiberlaubnis hat;
S_IREAD Wenn der Benutzer Leseerlaubnis hat.

Die Bit-Maske enthält auch Bits für die Schreib-/Leseerlaubnis; diese werden entsprechend dem Zugriffsmodus der Datei gesetzt.

Rückgabewert *fstat* und *stat* liefern den Wert 0 zurück, wenn sie die Informationen über die offene Datei erfolgreich abgerufen haben. Im Fall eines Fehlers ist der Rückgabewert -1 und die globale Variable *errno* wird auf folgenden Wert gesetzt:

EBADF Ungültiges Handle

Siehe auch *access*, *chmod*

Beispiel

```
#include <sys\stat.h>
#include <stdio.h>
#include <time.h>

int main(void)
{
   struct stat statbuf;
   FILE *stream;

   /* open a file for update */
   if ((stream = fopen("DUMMY.FIL", "w+"))
       == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return(1);
   }
   fprintf(stream, "This is a test");
   fflush(stream);

   /* get information about the file */
   fstat(fileno(stream), &statbuf);
   fclose(stream);

   /* display the information returned */
   if (statbuf.st_mode & S_IFCHR)
      printf("Handle refers to a device.\n");
   if (statbuf.st_mode & S_IFREG)
      printf("Handle refers to an ordinary file.\n");
   if (statbuf.st_mode & S_IREAD)
```

```
        printf("User has read permission on file.\n");
    if (statbuf.st_mode & S_IWRITE)
        printf("User has write permission on file.\n");

    printf("Drive letter of file: %c\n",
           'A'+statbuf.st_dev);
    printf("Size of file in bytes: %ld\n",
           statbuf.st_size);
    printf("Time file last opened: %s\n",
           ctime(&statbuf.st_ctime));
    return 0;
}
```

_fstr...

Vergleichen Sie die Beschreibung der folgenden Funktionen, die auch für die jeweilige *far*-Version Gültigkeit hat:

streat, strchr, strespn, strdup, stricmp, strlen, strlwr, strneat, strucmp, strncpy, strnicmp, strnset, strpbrk, strrchr, strrev, strset, strspn, strstr, strtok, strupr

ftell

Funktion — Liefert den aktuellen Dateizeiger.

Syntax

```
#include <stdio.h>
long int ftell(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *ftell* liefert den aktuellen Dateizeiger von *stream* zurück. Die Position wird relativ zum Dateianfang in Bytes gemessen (falls es sich um eine Binärdatei handelt).

Der von *ftell* zurückgelieferte Wert kann bei einem nachfolgenden Aufruf von *fseek* verwendet werden.

Rückgabewert — *ftell* liefert bei fehlerfreier Ausführung die aktuelle Position des Dateizeigers zurück. Im Fall eines Fehlers ist der Rückgabewert -1L und die globale Variable *errno* wird auf einen positiven Wert gesetzt.

Siehe auch — *fgetpos, fseek, fsetpos, lseek, rewind, tell*

Beispiel

```
#include <stdio.h>

int main(void)
{
```

```
   FILE *stream;

   stream = fopen("MYFILE.TXT", "w+");
   fprintf(stream, "This is a test");
   printf("The file pointer is at byte %ld\n",
          ftell(stream));
   fclose(stream);
   return 0;
}
```

ftime

Funktion — Speichert die Systemzeit in einer Struktur des Typs *timeb*.

Syntax

```
#include <sys\timeb.h>
void ftime(struct timeb *buf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *ftime* ist nur unter UNIX System V verfügbar. Die Funktion ermittelt die aktuelle Systemzeit und speichert sie in den Feldern der durch *buf* bezeichneten Struktur. Die Struktur *timeb* ist in sys\timeb.h folgendermaßen definiert;

```
struct timeb {
    long time;
    short millitm;
    short timezone;

    short dstflag;
};
```

time ist die Zeit in Sekunden seit dem 1.1.1970 00:00:00 GMT (Greenwich Mean Time), *millitm* der Bruchteil einer Sekunde in Millisekunden und *timezone* der lokale Zeitunterschied zur GMT in Minuten. Der Zeitunterschied zur GMT wird über die globale Variable *timezone* berechnet, deren Wert durch *tzset* gesetzt wird.

dstflag gibt an, ob in der aktuellen Zeitzone eine Verschiebung durch die Sommerzeit berücksichtigt wird.

Hinweis: ftime ruft *tzset* auf. Es ist daher nicht nötig, vor dem Gebrauch von *ftime tzset* explizit aufzurufen.

Rückgabewert — Keiner.

Siehe auch — *asctime*, *ctime*, *gmtime*, *localtime*, *stime*, *time*, *tzset*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>
#include <sys\timeb.h>

/* pacific standard & daylight savings */
char *tzstr = "TZ=PST8PDT";

int main(void)
{
   struct timeb t;
   putenv(tzstr);
   tzset();
   ftime(&t);
   printf("Seconds since 1/1/1970 GMT: %ld\n",t.time);
   printf("Thousandths of a second: %d\n", t.millitm);
   printf("Difference between local time and GMT: "
          "%d\n", t.timezone);
   printf("Daylight savings in effect (1) not (0): "
          "%d\n", t.dstflag);
   return 0;
}
```

_fullpath

Funktion

Konvertiert einen relativen Pfadnamen in einen absoluten.

Syntax

```
#include <stdlib.h>
char* _fullpath(char *buffer, const char *path,
                int buflen);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_fullpath konvertiert den relativen Pfadnamen in *path* in einen absoluten Pfadnamen. Dieser wird in dem Zeichen-Array gespeichert, auf das *buffer* zeigt. *buflen* gibt an, wieviele Zeichen in *buffer* gespeichert werden können. *_fullpath* liefert NULL zurück, wenn der Puffer nicht groß genug für den absoluten Pfadnamen ist oder der Pfad eine ungültige Laufwerksangabe enthält.

Wenn *buffer* NULL ist, belegt *_fullpath* einen Puffer mit bis zu _MAX_PATH Zeichen. Dieser Puffer sollte durch einen Aufruf von *free* freigegeben werden, wenn er nicht mehr benötigt wird. _MAX_PATH ist in stdlib.h definiert.

Rückgabewert

Bei fehlerfreier Ausführung liefert *_fullpath* einen Zeiger auf den Puffer zurück, der den absoluten Pfadnamen enthält. Andernfalls ist das Ergebnis NULL.

Siehe auch

_makepath, _splitpath

Beispiel

```
 #include <stdio.h>
#include <stdlib.h>

char buf[_MAX_PATH];

void main(int argc, char *argv[])
{
  for ( ; argc; argv++, argc--)
  {
    if (_fullpath(buf, argv[0], _MAX_PATH) == NULL)
      printf("Unable to obtain full path of "
             "%s\n", argv[0]);
    else
      printf("Full path of %s is %s\n",argv[0],buf);
  }
}
```

fwrite

Funktion Schreibt Daten in einen Stream.

Syntax

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t n,
              FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *fwrite* fügt *n* Datenelemente mit jeweils *size* Bytes (die aus einem Speicherbereich gelesen werden, dessen Startadresse durch *ptr* gegeben ist) an die gegebene Datei an. Dabei werden *n * size* Bytes geschrieben.
ptr kann auf beliebige Objekte zeigen.

Rückgabewert *fwrite* liefert bei fehlerfreier Ausführung die Anzahl der geschriebenen Datenelemente (nicht Bytes) zurück. Im Fall eines Fehlers wird die Anzahl von Datenelementen zurückgeliefert, die noch fehlerfrei geschrieben werden konnte.

Siehe auch *fopen, fread*

Beispiel

```
#include <stdio.h>

struct mystruct
{
  int i;
  char ch;
};
```

```
int main(void)
{
   FILE *stream;
   struct mystruct s;

   if ((stream = fopen("TEST.$$$", "wb")) == NULL)
          /* open file TEST.$$$ */
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
   s.i = 0;
   s.ch = 'A';
   fwrite(&s, sizeof(s), 1, stream);
          /* write struct s to file */
   fclose(stream); /* close file */
   return 0;
}
```

gcvt

Funktion — Konvertiert eine Gleitkommazahl in einen String.

Syntax

```
#include <stdlib.h>
char *gcvt(double value, int ndec, char *buf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *gcvt* konvertiert *value* in einen nullterminierten ASCII-String, der in *buf* gespeichert wird. Die Funktion versucht, *ndec* signifikante Ziffern im F-Format von FORTRAN zu erzeugen. Wenn dies nicht möglich ist, liefert sie den Wert im E-Format von *printf* (druckfertig formatiert). Abschließende Nullen werden möglicherweise unterdrückt.

Rückgabewert — *gcvt* liefert die Adresse des durch *buf* bezeichneten Strings zurück.

Siehe auch — *ecvt*, *fcvt*, *sprintf*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   char str[25];
   double num;
   int sig = 5; /* significant digits */
```

```
   /* a regular number */
   num = 9.876;
   gcvt(num, sig, str);
   printf("string = %s\n", str);

   /* a negative number */
   num = -123.4567;
   gcvt(num, sig, str);
   printf("string = %s\n", str);

   /* scientific notation */
   num = 0.678e5;
   gcvt(num, sig, str);
   printf("string = %s\n", str);

   return(0);
}
```

geninterrupt

Funktion — Erzeugt einen Software-Interrupt.

Syntax

```
#include <dos.h>
void geninterrupt(int intr_num);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung — Das Makro *geninterrupt* löst einen Software-Interrupt für den Interrupt *intr_num* aus. Der Zustand der Register nach dem Aufruf hängt von dem aufgerufenen Interrupt ab.

Hinweis: Interrupts können Register, die von C verwendet werden, in einem nicht vorhersehbaren Zustand zurücklassen.

Rückgabewert — Keiner.

Siehe auch — *bdos, bdosptr, disable, enable, getvect, int86, int86x, intdos, intdosx, intr*

Beispiel

```
#include <conio.h>
#include <dos.h>

/* function prototype */
void writechar(char ch);

int main(void)
{
   clrscr();
   gotoxy(80,25);
```

```
   writechar('*');
   getch();
   return 0;
}

/*
   outputs a character at the current cursor
   position using the video BIOS to avoid
   the scrolling of the screen when writing
   to location (80,25).
*/

void writechar(char ch)
{
   struct text_info ti;
   /* grab current text settings */
   gettextinfo(&ti);
   /* interrupt 0x10 sub-function 9 */
   _AH = 9;
   /* character to be output */
   _AL = ch;
   _BH = 0;                    /* video page */
   _BL = ti.attribute;  /* video attribute */
   _CX = 1;           /* repetition factor */
   geninterrupt(0x10);  /* output the char */
}
```

getarccoords

Funktion Ermittelt die Koordinaten des letzten Aufrufs von *arc*.

Syntax

```
#include <graphics.h>
void far getarccoords(struct arccoordstype
                      far *arccoords);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getarccoords* liefert die Daten des letzten Aufrufs von *arc* in einer Struktur des Typs *arccoordstype* zurück, die in graphics.h folgendermaßen definiert ist:

```
struct arccoordstype {
   int x, y;
   int xstart, ystart, xend, yend;
};
```

Die Elemente dieser Struktur geben den Mittelpunkt (*x,y*), den Startpunkt (*xstart, ystart*) und den Endpunkt (*xend, yend*)

des Bogens an. Diese Werte werden benötigt, um einen Kreisbogen z.B. mit einer Linie fortzusetzen.

Rückgabewert Keiner.

Siehe auch *arc, fillellipse, sector*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct arccoordstype arcinfo;
   int midx, midy;
   int stangle = 45, endangle = 270;
   char sstr[80], estr[80];

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

/* draw arc and get coordinates */
   setcolor(getmaxcolor());
   arc(midx, midy, stangle, endangle, 100);
   getarccoords(&arcinfo);

/* convert arc information into strings */
   sprintf(sstr, "*- (%d, %d)", arcinfo.xstart,
           arcinfo.ystart);
   sprintf(estr, "*- (%d, %d)", arcinfo.xend,
           arcinfo.yend);

   /* output the arc information */
   outtextxy(arcinfo.xstart,
```

```
                arcinfo.ystart, sstr);
   outtextxy(arcinfo.xend,
                arcinfo.yend, estr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getaspectratio

Funktion — Ermittelt das aktuelle physikalische Höhen-/ Seitenverhältnis des Bildschirms.

Syntax

```
#include <graphics.h>
void far getaspectratio(int far *xasp, int far *yasp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Der horizontale Faktor **yasp* ist auf 10000 normiert. Für alle Grafikadapter (mit Ausnahme von VGA) ist **xasp* kleiner als **yasp*, weil die vertikale Ausdehnung der Pixel kleiner als die horizontale Ausdehnung ist. Bei VGA-Karten sind die Pixel quadratisch, d.h. **xasp* = **yasp*.

Allgemein kann das Verhältnis zwischen **yasp* und **xasp* folgendermaßen dargestellt werden:

$$*yasp = 10000$$
$$*xasp <= 10000$$

getaspectratio ermittelt die Werte in **xasp* und **yasp*.

Rückgabewert — Keiner.

Siehe auch — *arc, circle, ellipse, fillellipse, pieslice, sector, setaspectratio*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xasp, yasp, midx, midy;

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
```

```
/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

/* get current aspect ratio settings */
   getaspectratio(&xasp, &yasp);

/* draw normal circle */
   circle(midx, midy, 100);
   getch();

/* draw wide circle */
   cleardevice();
   setaspectratio(xasp/2, yasp);
   circle(midx, midy, 100);
   getch();

/* draw narrow circle */
   cleardevice();
   setaspectratio(xasp, yasp/2);
   circle(midx, midy, 100);

/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getbkcolor

Funktion Liefert die aktuelle Hintergrundfarbe zurück.

Syntax

```
#include <graphics.h>
int far getbkcolor(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getbkcolor* liefert die aktuelle Hintergrundfarbe. Details finden Sie unter *setbkcolor*.

Rückgabewert *getbkcolor* liefert die aktuelle Hintergrundfarbe zurück.

Siehe auch *getcolor, getmaxcolor, getpalette, setbkcolor*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int bkcolor, midx, midy;
   char bkname[35];

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

/* for centering text on the display */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

/* get the current background color */
   bkcolor = getbkcolor();

/* convert color value into a string */
   itoa(bkcolor, bkname, 10);
   strcat(bkname," is the current background color.");

/* display a message */
   outtextxy(midx, midy, bkname);
```

```
/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getc

Funktion Liest ein Zeichen aus einem Stream.

Syntax
```
#include <stdio.h>
int getc(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *getc* ist ein Makro, das das jeweils nächste Zeichen aus dem angegebenen Eingabe-Stream liest, zurückliefert und den zugehörigen Dateizeiger entsprechend inkrementiert.

Rückgabewert *getc* liefert bei fehlerfreier Ausführung das gelesene Zeichen als **int** ohne Vorzeichenerweiterung zurück. Am Dateiende oder im Fall eines Fehlers liefert die Funktion EOF zurück.

Siehe auch *fgetc*, *getch*, *getchar*, *getche*, *gets*, *putc*, *putchar*, *ungetc*

Beispiel
```
#include <stdio.h>

int main(void)
{
   char ch;

   printf("Input a character:");
/* read a character from the
   standard input stream */
   ch = getc(stdin);
   printf("The character input was: '%c'\n", ch);
   return 0;
}
```

getcbrk

Funktion Ermittelt, ob DOS auf Strg Untbr prüft.

Syntax
```
#include <dos.h>
int getcbrk(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *getcbrk* verwendet den DOS-Systemaufruf 0x33, um zu ermitteln, wann DOS auf Strg Untbr prüft.

Rückgabewert *getcbrk* liefert 0, wenn die Überprüfung abgeschaltet ist, ansonsten den Wert 1 zurück.

Siehe auch *ctrlbrk*, *setcbrk*

Beispiel
```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   if (getcbrk())
      printf("Cntrl-brk flag is on\n");
   else
      printf("Cntrl-brk flag is off\n");

   return 0;
}
```

getch

Funktion Liest ein Zeichen von der Tastatur, ohne es auf dem Bildschirm auszugeben.

Syntax
```
#include <conio.h>
int getch(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getch* liest ein einzelnes Zeichen von der Tastatur, ohne es als »Echo« am Bildschirm auszugeben.

Rückgabewert *getch* liefert das von der Tastatur gelesene Zeichen zurück.

Siehe auch *cgets*, *cscanf*, *fgetc*, *getc*, *getchar*, *getche*, *getpass*, *kbhit*, *putch*, *ungetch*

Beispiel
```
#include <conio.h>
#include <stdio.h>

int main(void)
{
  int c;
  int extended = 0;
  c = getch();
  if (!c)
    extended = getch();
  if (extended)
    printf("The character is extended\n");
```

```
  else
    printf("The character isn't extended\n");

  return 0;
}
```

getchar

Funktion — Liest ein Zeichen aus *stdin*.

Syntax

```
#include <stdio.h>
int getchar(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung — *getchar* ist ein Makro, das das jeweils nächste Zeichen aus dem Eingabe-Stream *stdin* liest und zurückliefert. Es ist definiert als *getc*(*stdin*).

Rückgabewert — *getchar* liefert bei fehlerfreier Ausführung das gelesene Zeichen als **int**-Wert ohne Vorzeichenerweiterung zurück. Am Dateiende oder im Fall eines Fehlers liefert die Funktion EOF zurück.

Siehe auch — *fgetc*, *fgetchar*, *getc*, *getch*, *getche*, *gets*, *putc*, *putchar*, *scanf*, *ungetc*

Beispiel

```
#include <stdio.h>

int main(void)
{
   int c;

   /* Note that getchar reads from stdin and
      is line buffered; this means it will
      not return until you press ENTER. */

   while ((c = getchar()) != '\n')
      printf("%c", c);

   return 0;
}
```

getche

Funktion — Liest ein Zeichen von der Tastatur und gibt es auf dem Bildschirm aus.

Syntax

```
#include <conio.h>
int getche(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getche* liest ein einzelnes Zeichen von der Tastatur und gibt dieses Zeichen auf dem Bildschirm in das aktuelle Textfenster aus. *getche* schreibt direkt in den Bildschirmspeicher oder benutzt BIOS-Routinen.

Rückgabewert *getche* liefert das von der Tastatur gelesene Zeichen zurück.

Siehe auch *cgets*, *cscanf*, *fgetc*, *getc*, *getch*, *getchar*, *kbhit*, *putch*, *ungetch*

Beispiel

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
   char ch;

   printf("Input a character:");
   ch = getche();
   printf("\nYou input a '%c'\n", ch);
   return 0;
}
```

getcolor

Funktion Liefert die aktuelle Zeichenfarbe zurück.

Syntax

```
#include <graphics.h>
int far getcolor(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Die Zeichenfarbe ist der Wert, auf den Pixel gesetzt werden, wenn Linien und Umrisse gezeichnet werden.

Ein Beispiel: Im Modus CGAC0 enthält die Farbpalette vier Farben: die Hintergrundfarbe, hellgrün, hellrot und gelb. Wenn *getcolor* in diesem Modus den Wert 1 zurückliefert, wird hellgrün als Zeichenfarbe benutzt.

Rückgabewert *getcolor* liefert die aktuelle Zeichenfarbe zurück.

Siehe auch *getbkcolor*, *getmaxcolor*, *getpalette*, *setcolor*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void)
{
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int color, midx, midy;
   char colname[35];

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

/* for centering text on the display */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

/* get the current drawing color */
   color = getcolor();

/* convert color value into a string */
   itoa(color, colname, 10);
   strcat(colname,
          " is the current drawing color.");

/* display a message */
   outtextxy(midx, midy, colname);

/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getcurdir

Funktion — Ermittelt das aktuelle Verzeichnis eines Laufwerks.

Syntax

```
#include <dir.h>
int getcurdir(int drive, char *directory);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *getcurdir* ermittelt das aktuelle Verzeichnis auf dem durch *drive* angegebenen Laufwerk.

Wenn *drive* den Wert 0 hat, wird das aktuelle Laufwerk verwendet, ansonsten gilt A = 1, B = 2 usw.

directory zeigt auf einen Speicherbereich der Größe MAXDIR – hier trägt *getcurdir* den nullterminierten Namen des ermittelten Verzeichnisses ein (ohne führenden Backslash und ohne Laufwerksbezeichner).

Rückgabewert — Bei fehlerfreier Ausführung liefert *getcurdir* den Wert 0, im Fall eines Fehlers -1 zurück.

Siehe auch — *chdir*, *getcwd*, *getdisk*, *mkdir*, *rmdir*

Beispiel

```
#include <dir.h>
#include <stdio.h>
#include <string.h>

char *current_directory(char *path)
{
   strcpy(path, "X:\\");
      /* fill string with form of response: X:\ */
   path[0] = 'A' + getdisk();
      /* replace X with current drive letter */
   getcurdir(0, path+3);
      /* fill rest of string with current directory */
   return(path);
}

int main(void)
{
   char curdir[MAXPATH];

   current_directory(curdir);
   printf("The current directory is %s\n", curdir);

   return 0;
}
```

getcwd

Funktion — Liefert das aktuelle Arbeitsverzeichnis.

Syntax

```
#include <dir.h>
char *getcwd(char *buf, int buflen);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *getcwd* liefert den Pfadnamen des aktuellen Arbeitsverzeichnisses (inklusive des Laufwerks) als String in *buf* zurück, dessen Maximallänge durch den Parameter *buflen* gesetzt wird.

Ist der vollständige Pfadname einschließlich des abschließenden Nullzeichens länger als *buflen* Bytes, so wird ein Fehler erzeugt.

Wenn der als *buf* übergebene Zeiger den Wert NULL hat, belegt *getcwd* über einen Aufruf von *malloc* selbständig einen Bereich von *buflen* Bytes. Später kann dieser automatisch belegte Speicherbereich mit *free* freigegeben werden.

Rückgabewert — *getcwd* liefert folgende Werte zurück:

- Ist *buf* bei der Eingabe ungleich NULL, wird bei fehlerfreier Ausführung *buf*, im Fall eines Fehlers NULL zurückgeliefert.
- Ist *buf* bei der Eingabe gleich NULL, liefert *getcwd* bei fehlerfreier Ausführung einen Zeiger auf den belegten Puffer zurück.
- Im Fall eines Fehlers wird *errno* auf einen der folgenden Werte gesetzt:

ENODEV	Gerät nicht bereit/nicht vorhanden
ENOMEM	Nicht genug Platz im Hauptspeicher
ERANGE	Ergebnis außerhalb des Bereichs

Siehe auch — *chdir, getcurdir, _getdcwd, _getdisk, mkdir, rmdir*

Beispiel

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
   char buffer[MAXPATH];

   getcwd(buffer, MAXPATH);
   printf("The current directory is: %s\n", buffer);
   return 0;
}
```

getdate, _dos_getdate, _dos_setdate, setdate

Funktion

Lesen und Setzen des Systemdatums.

Syntax

```
#include <dos.h>
void getdate(struct date *datep);
void _dos_getdate(struct dosdate_t *datep);
void setdate(struct date *datep);
unsigned _dos_setdate(struct dosdate_t *datep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

getdate füllt die Felder der *date*-Struktur, auf die *datep* zeigt, mit dem aktuellen Datum des Systems.
setdate setzt das Systemdatum (Tag, Monat, und Jahr) auf das Datum in der über *datep* angegebenen *date*-Struktur.

Der Strukturtyp *date* ist folgendermaßen definiert:

```
struct date {
   int da_year;   /* Aktuelles Jahr */
   char da_day;   /* Tag im Monat (1 bis 31) */
   char da_mon;   /* Monat (1 = Januar) */
};
```

_dos_getdate füllt die Felder der *dosdate_t*-Struktur, auf die *datep* zeigt, mit dem aktuellen Systemdatum.

Der Strukturtyp *dosdate_t* ist folgendermaßen definiert:

```
struct dosdate_t {
   unsigned char day;        /* 1 bis 31 */
   unsigned char month;      /* 1 bis 12 */
   unsigned int year;        /* 1980 bis 2099 */
   unsigned char dayofweek;  /* 0 bis 6
                             /* (0 = Sonntag) */
};
```

Rückgabewert

_dos_getdate, *getdate* und *setdate* liefern keinen Wert zurück. Wurde das Datum erfolgreich gesetzt, liefert *_dos_setdate* den Wert 0 zurück. Andernfalls liefert die Funktion einen Wert ungleich Null und die globale Variable *errno* bekommt den Wert:

EINVAL Ungültiges Datum

Siehe auch

ctime, *gettime*, *settime*

Beispiel

```
#include <dos.h>
#include <stdio.h

int main(void)
```

```
{
   struct date d;

   getdate(&d);
   printf("The current year is: %d\n", d.da_year);
   printf("The current day is: %d\n", d.da_day);
   printf("The current month is: %d\n", d.da_mon);
   return 0;
}
```

_getdcwd

Funktion

Liefert das aktuelle Arbeitsverzeichnis des angegebenen Laufwerks.

Syntax

```
#include <direct.h>
char *_getdcwd(int drive, char *buffer, int buflen);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_getdcwd liefert den vollständigen Pfadnamen des Arbeitsverzeichnisses im angegebenen Laufwerk (inklusive des Laufwerknamens) als String in *buffer* zurück, dessen Maximallänge durch den Parameter *buflen* gesetzt wird. Ist der vollständige Pfadname einschließlich des abschließenden Nullzeichens länger als *buflen* Bytes, so wird ein Fehler erzeugt. Für den Parameter *drive* gilt: 0 = aktuelles Laufwerk, 1 = A, 2 = B, etc.

Wenn der als *buffer* übergebene Zeiger den Wert NULL hat, belegt *_getdcwd* einen Puffer von mindestens *buflen* Bytes. Später kann dieser Speicherbereich durch Übergabe des Rückgabewerts von *_getdcwd* an *free* wieder freigegeben werden.

Rückgabewert

_getdcwd liefert bei fehlerfreier Ausführung einen Zeiger auf den Puffer zurück, der den Namen des aktuellen Verzeichnisses im angegebenen Laufwerk enthält. Im Falle eines Fehlers liefert *_getdcwd* den Wert 0 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

ENOMEM	Nicht genug Platz im Hauptspeicher (*buffer* ist NULL)
ERANGE	Verzeichnisname ist länger als *buflen* (*buffer* ist ungleich NULL)

Siehe auch

chdir, *getcwd*, *_getdrive*, *mkdir*, *rmdir*

Beispiel

```
#include <direct.h>
#include <stdio.h>

char buf[65];
void main()
```

```
{
  if (_getdcwd(3, buf, sizeof(buf)) == NULL)
    perror("Unable to get current directory "
           "of drive C");
  else
    printf("Current directory of drive C is %s\n",
           buf);
}
```

getdefaultpalette

Funktion Liefert einen Zeiger auf die Standardfarbpalette zurück.

Syntax
```
#include <graphics.h>
struct palettetype *far getdefaultpalette(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getdefaultpalette* sucht eine Struktur des Typs *palettetype*. Diese Struktur enthält die Farbpalette, die bei der Initialisierung durch den Treiber mit *initgraph* gesetzt wurde.

Rückgabewert *getdefaultpalette* liefert einen Zeiger auf die Farbpalette zurück, die bei der Initialisierung des aktuellen Treibers gesetzt wurde.

Siehe auch *getpalette, initgraph*

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int i;

/* structure for returning palette copy */
   struct palettetype far *pal=(void *) 0;

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
```

```
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   setcolor(getmaxcolor());

/* return a pointer to the default palette */
   pal = getdefaultpalette();

   for (i=0; i<16; i++)
   {
      printf("colors[%d] = %d\n", i, pal->colors[i]);
      getch();
   }

/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getdfree

Funktion

Ermittelt den freien Platz auf einer Diskette/Festplatte.

Syntax

```
#include <dos.h>
void getdfree(unsigned char drive,
              struct dfree *dtable);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

getdfree ermittelt Informationen über den freien Platz auf dem durch *drive* angegebenen Laufwerk und speichert sie in einer Struktur des Typs *dfree*.

Wenn *drive* den Wert 0 hat, wird das aktuelle Laufwerk bearbeitet, ansonsten gilt A = 1, B = 2 usw.

Der Strukturtyp *dfree* ist folgendermaßen definiert:

```
struct dfree {
   unsigned df_avail;  /* freie Cluster */
   unsigned df_total;  /* Gesamtzahl Cluster */
   unsigned df_bsec;   /* Byte pro Sektor */
   unsigned df_sclus;  /* Sektoren pro Cluster */
};
```

Rückgabewert

getdfree liefert keinen Wert zurück. Im Fall eines Fehlers bekommt das Element *df_sclus* den Wert 0xFFFF zugewiesen.

Siehe auch *getfat, getfatd*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <dos.h>

int main(void)
{
   struct dfree free;
   long avail;
   int drive;

   drive = getdisk();
   getdfree(drive+1, &free);
   if (free.df_sclus == 0xFFFF)
   {
      printf("Error in getdfree() call\n");
      exit(1);
   }

   avail =  (long) free.df_avail
            * (long) free.df_bsec
            * (long) free.df_sclus;
   printf("Drive %c: has %ld bytes available\n",
          'A' + drive, avail);

   return 0;
}
```

getdisk, setdisk

Funktion Ermittelt oder setzt die aktuelle Laufwerksnummer.

Syntax

```
#include <dir.h>
int getdisk(void);
int setdisk(int drive);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *getdisk* liefert die Nummer des aktuellen Laufwerks als Integer zurück: 0 entspricht Laufwerk A, 1 entspricht Laufwerk B usw. (entspricht der DOS-Funktion 0x19).

setdisk setzt das aktuelle Laufwerk auf die Nummer in *drive*: 0 entspricht Laufwerk A, 1 entspricht Laufwerk B usw. (entspricht der DOS-Funktion 0x0E).

Rückgabewert *getdisk* liefert die aktuelle Laufwerksnummer zurück. *setdisk* liefert die Anzahl der verfügbaren Laufwerke zurück.

Siehe auch *getcurdir, getcwd*

Beispiel

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
  int disk, maxdrives = setdisk(2);
  disk = getdisk() + 'A';
  printf("\nThe number of logical drives is:%d\n",
         maxdrives);
  printf("The current drive is: %c\n", disk);
  return 0;
}
```

_getdrive

Funtion Ermittelt die aktuelle Laufwerksnummer.

Syntax

```
#include <direct.h>
int _getdrive(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung *_getdrive* verwendet die DOS-Funktion 0x19 zur Ermittlung der aktuellen Laufwerksnummer. Die Funktion liefert einen Integerwert zurück, wobei gilt: 1 =A, 2 =B, 3 =C, usw.

Rückgabewert *_getdrive* liefert die aktuelle Laufwerksnummer zurück.

Siehe auch *_dos_getdrive, _dos_setdrive, _getdcwd*

Beispiel

```
#include <stdio.h>
#include <direct.h>

int main(void)
{
  int disk;
  disk = _getdrive() + 'A' - 1;
  printf("The current drive is: %c\n", disk);
  return 0;
}
```

getdrivername

Funktion Liefert einen Zeiger auf einen String zurück, der den Namen des aktuellen Grafiktreibers enthält.

Syntax

```
#include <graphics.h>
char *far getdrivername(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
	x			

Beschreibung Nach der Initialisierung des Grafikpakets mit *initgraph* liefert *getdrivername* einen Zeiger auf den Namen des aktuell geladenen Grafiktreibers.

Rückgabewert *getdrivername* liefert einen Zeiger auf einen String zurück, der den Namen des aktuell geladenen Grafiktreibers enthält.

Siehe auch *initgraph*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

/* stores the device driver name */
   char *drivername;

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   setcolor(getmaxcolor());

/* get name of the device driver in use */
   drivername = getdrivername();

/* for centering text on the screen */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

/* output the name of the driver */
   outtextxy(getmaxx() / 2, getmaxy() / 2,
             drivername);
```

```
/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getdta

Funktion — Liefert die Adresse der aktuellen DTA zurück.

Syntax

```
#include <dos.h>
char far *getdta(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *getdta* liefert die aktuelle *disk transfer address* (DTA).

In den Modellen *small* und *medium* geht Borland C++davon aus, daß sich die DTA im aktuellen Datensegment befindet. Solange ausschließlich mit C-Modulen gearbeitet wird, ist das auch der Fall – von einem Maschinenprogramm aus kann die DTA aber auf eine beliebige Adresse gesetzt werden.

In den Modellen *compact*, *large* und *huge* wird mit vollständigen Adressen gearbeitet – hier kann sich die DTA auch außerhalb der Programmsegmente befinden.

Rückgabewert — *getdta* liefert einen Zeiger auf die aktuelle DTA zurück.

Siehe auch — *fcb* (Struktur), *setdta*

Beispiel

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
   char far *dta;

   dta = getdta();
   printf("The current disk transfer address "
          "is: %Fp\n", dta);
   return 0;
}
```

getenv

Funktion — Liest einen String aus der Tabelle *Environment*.

Syntax

```
#include <stdlib.h>
char *getenv(const char *name);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *getenv* sucht die Tabelle nach dem über *name* angegebenen Eintrag ab. Unter DOS muß *name* in Großbuchstaben vorliegen, in anderen Systemen kann er entweder in Groß- oder Kleinbuchstaben geschrieben sein. *name* darf kein Gleichheitszeichen (=) enthalten. Falls kein Eintrag des Namens *name* gefunden werden konnte, liefert *getenv* einen NULL-Zeiger.

Rückgabewert Bei fehlerfreier Ausführung liefert *getenv* den Wert zurück, der der Variablen zugeordnet ist. Falls kein Eintrag des Namens *name* gefunden werden konnte, liefert *getenv* einen NULL-Zeiger.

Hinweis: Environment-Einträge dürfen nicht direkt geändert werden. Wenn ein bestimmter Wert geändert werden soll, muß *putenv* verwendet werden.

Siehe auch *environ* (globale Variable), *getpsp*, *putenv*

Beispiel
```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char *s;

    s=getenv("COMSPEC");
/* get the comspec environment parameter */
    printf("Command processor: %s\n",s);
/* display comspec parameter */

    return 0;
}
```

getfat

Funktion Ermittelt Informationen über die FAT des angegebenen Laufwerks.

Syntax
```
#include <dos.h>
void getfat(unsigned char drive,
            struct fatinfo *dtable);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *getfat* füllt die mit *dtable* angegebene Struktur des Typs *fatinfo* mit Informationen über die FAT (File Allocation Table) des Laufwerks *drive*.

Wenn *drive* den Wert 0 hat, wird das aktuelle Laufwerk benutzt, ansonsten gilt A = 1, B = 2 usw. Der Strukturtyp *fatinfo* ist folgendermaßen definiert:

```
struct fatinfo {
   char fi_sclus;       /* Sektoren pro Cluster */
   char fi_fatid;       /* FAT ID-Byte */
   unsigned fi_nclus;   /* Gesamtzahl Cluster */
   int fi_bysec;        /* Bytes pro Sektor */
};
```

Rückgabewert Keiner.

Siehe auch *getdfree, getfatd*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   struct fatinfo diskinfo;
   int flag = 0;

   printf("Please insert disk in drive A\n");
   getchar();

   getfat(1, &diskinfo);
/* get drive information */

   printf("\nDrive A: is ");
   switch((unsigned char) diskinfo.fi_fatid)
   {
      case 0xFD:
        printf("360K low density\n");
        break;

      case 0xF9:
        printf("1.2 Meg high density\n");
        break;

      default:
        printf("unformatted\n");
        flag = 1;
   }

   if (!flag)
   {
      printf("  sectors per cluster %5d\n",
             diskinfo.fi_sclus);
      printf("   number of clusters %5d\n",
             diskinfo.fi_nclus);
      printf("     bytes per sector %     5d\n",
```

```
                diskinfo.fi_bysec);
    }

    return 0;
}
```

getfatd

Funktion: Ermittelt Informationen über die FAT des aktuellen Laufwerks.

Syntax:

```
#include <dos.h>
void getfatd(struct fatinfo *dtable);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung: *getfatd* füllt die über *dtable* angegebene Struktur des Typs *fatinfo* mit Informationen über die FAT des Datenträgers im aktuellen Laufwerk. Der Strukturtyp *fatinfo* ist folgendermaßen definiert:

```
struct fatinfo {
    char fi_sclus;    /* Sektoren pro Cluster */
    char fi_fatid;    /* FAT ID-Byte */
    int fi_nclus;     /* Gesamtzahl Cluster */
    int fi_bysec;     /* Bytes pro Sektor */
};
```

Rückgabewert: Keiner.

Siehe auch: *getdfree, getfat*

Beispiel:

```
#include <stdio.h>
#include <dos.h>

int main()
{
    struct fatinfo diskinfo;

    /* get default drive information */
    getfatd(&diskinfo);
    printf("\nDefault Drive:\n");
    printf("sectors per cluster: %5d\n",
           diskinfo.fi_sclus);
    printf("FAT ID byte:         %5X\n",
           diskinfo.fi_fatid & 0xFF);
    printf("number of clusters   %5d\n",
           diskinfo.fi_nclus);
    printf("bytes per sector     %5d\n",
           diskinfo.fi_bysec);
    return 0;
}
```

getfillpattern

Funktion

Kopiert das benutzerdefinierte Füllmuster in den Speicher.

Syntax

```
#include <graphics.h>
void far getfillpattern(char far *pattern);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

getfillpattern kopiert das mit *setfillpattern* vom Benutzer definierte Füllmuster in den 8-Byte-Bereich, auf den *pattern* zeigt.

pattern ist ein Zeiger auf eine Folge von 8 Bytes, bei dem jedes Byte für 8 Pixel des Musters steht. Für jedes Bit, das auf 1 gesetzt ist, wird das korrespondierende Pixel gezeichnet. Zum Beispiel stellt das folgende benutzerdefinierte Füllmuster ein Schachbrett dar:

```
char checkboard[8] = {
   0xAA, 0x55, 0xAA, 0x55, 0xAA, 0xAA, 0x55
};
```

Rückgabewert

Keiner.

Siehe auch

getfillsettings, setfillpattern

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;
   char pattern[8] = {0x00, 0x70, 0x20, 0x27,
                      0x25, 0x27, 0x04, 0x04};

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
```

```
    maxx = getmaxx();
    maxy = getmaxy();
    setcolor(getmaxcolor());

    /* select a user defined fill pattern */
    setfillpattern(pattern, getmaxcolor());

    /* fill the screen with the pattern */
    bar(0, 0, maxx, maxy);

    getch();

    /* get the current user defined fill pattern */
    getfillpattern(pattern);

    /* alter the pattern we grabbed */
    pattern[4] -= 1;
    pattern[5] -= 3;
    pattern[6] += 3;
    pattern[7] -= 4;

    /* select our new pattern */
    setfillpattern(pattern, getmaxcolor());

    /* fill the screen with the new pattern */
    bar(0, 0, maxx, maxy);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getfillsettings

Funktion

Ermittelt Informationen über das aktuelle Füllmuster und die für Flächenfüllungen verwendete Farbe.

Syntax

```
#include <graphics.h>
void far getfillsettings(struct fillsettingstype
                         far *fillinfo);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

getfillsettings ermittelt das aktuelle Füllmuster und die aktuelle Füllfarbe und speichert diese Informationen in der vom Zeiger *fillinfo* bezeichneten Struktur vom Typ *fillsettingstype*.

Die Struktur des Typs *fillsettingstype* ist in graphics.h folgendermaßen definiert:

```
struct fillsettingstype {
   int pattern;      /* aktuelles Füllmuster */
   int color;        /* aktuelle Füllfarbe */
};
```

Die Funktionen *bar*, *bar3d*, *fillpoly*, *floodfill* und *pieslice* füllen einen Bereich mit dem jeweils aktuellen Füllmuster in der aktuellen Füllfarbe. Es gibt 11 vordefinierte Füllmuster (wie Vollfarbe, Kreuzschraffur, Punkte, usw.). Symbolische Namen für die vordefinierten Muster werden durch den Aufzählungstyp *fill_patterns* in graphics.h zur Verfügung gestellt (siehe folgende Tabelle). Zusätzlich können Sie eigene Füllmuster definieren.

Hinweis: Wenn *pattern* gleich 12 (USER_FILL) ist, wird ein benutzerdefiniertes Füllmuster verwendet, ansonsten liefert *pattern* die Nummer eines vordefinierten Musters.

Der Aufzählungstyp *fill_patterns* – definiert in graphics.h – enthält Namen für die vordefinierten Füllmuster sowie einen Indikator für ein benutzerdefiniertes Muster.

Name	Wert	Füllmuster
EMPTY_FILL	0	Vollständig mit der Hintergrundfarbe
SOLID_FILL	1	Vollständig mit der Füllfarbe
LINE_FILL	2	----------
LTSLASH_FILL	3	/ / / /
SLASH_FILL	4	/ / / / (dicke Linien)
BKSLASH_FILL	5	\ \ \ \ (dicke Linien)
LTBKSLASH_FILL	6	\ \ \ \
HATCH_FILL	7	leicht schraffiert
XHATCH_FILL	8	stark schraffiert, überkreuzend
INTERLEAVE_FILL	9	abwechselnde Linien
WIDE_DOT_FILL	10	Punkte, weit auseinander
CLOSE_DOT_FILL	11	Punkte, dicht nebeneinander
USER_FILL	12	benutzerdefiniertes Füllmuster

Alle Füllmuster außer EMPTY_FILL füllen mit der aktuellen Füllfarbe; EMPTY_FILL verwendet die aktuelle Hintergrundfarbe.

Rückgabewert Keiner.

Siehe auch *getfillpattern*, *setfillpattern*, *setfillstyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL",
```

```
                    "SOLID_FILL",
                    "LINE_FILL",
                    "LTSLASH_FILL",
                    "SLASH_FILL",
                    "BKSLASH_FILL",
                    "LTBKSLASH_FILL",
                    "HATCH_FILL",
                    "XHATCH_FILL",
                    "INTERLEAVE_FILL",
                    "WIDE_DOT_FILL",
                    "CLOSE_DOT_FILL",
                    "USER_FILL"
                };

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct fillsettingstype fillinfo;
   int midx, midy;
   char patstr[40], colstr[40];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get information about current fill pattern
      and color                                    */
   getfillsettings(&fillinfo);

   /* convert fill information into strings */
   sprintf(patstr, "%s is the fill style.",
           fname[fillinfo.pattern]);
   sprintf(colstr, "%d is the fill color.",
           fillinfo.color);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
```

```
   outtextxy(midx, midy, patstr);
   outtextxy(midx, midy+2*textheight("W"), colstr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getftime, setftime

Funktion — Lesen und setzen Datum und Uhrzeit einer Datei.

Syntax

```
#include <io.h>
int getftime(int handle, struct ftime *ftimep);
int setftime(int handle, struct ftime *ftimep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *getftime* ermittelt Datum und Uhrzeit der durch *handle* bezeichneten offenen Datei und speichert die entsprechenden Werte in der durch *ftimep* angegebenen Struktur *ftime*.

setftime setzt Datum und Uhrzeit der durch *handle* bezeichneten offenen Datei auf die Werte der Struktur *ftime*, auf die *ftimep* zeigt. Nach dem Aufruf von *setftime* darf nicht in die Datei geschrieben werden – die geänderten Daten gehen dadurch verloren. Der Strukturtyp *ftime* ist folgendermaßen definiert:

```
struct ftime {
   unsigned ft_tsec: 5;     /* Zwei Sekunden */
   unsigned ft_min: 6;      /* Minuten */
   unsigned ft_hour: 5;     /* Stunden */
   unsigned ft_day: 5;      /* Tage */
   unsigned ft_month: 4;    /* Monate */
   unsigned ft_year: 7;     /* Jahr - 1980 */
 };
```

Rückgabewert — *getftime* und *setftime* liefern bei fehlerfreier Ausführung den Wert 0. Im Fehlerfall wird der Wert -1 zurückgeliefert und *errno* bekommt einen der folgenden Werte:

EINVFNC	Ungültige Funktionsnummer
EBADF	Ungültiges Handle

Siehe auch — *fflush, open, setftime*

Beispiel

```
#include <stdio.h>
#include <io.h>
```

```
int main(void)
{
   FILE *stream;
   struct ftime ft;

   if ((stream = fopen("TEST.$$$",
        "wt")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
   getftime(fileno(stream), &ft);
   printf("File time: %u:%u:%u\n",
          ft.ft_hour, ft.ft_min,
          ft.ft_tsec * 2);
   printf("File date: %u/%u/%u\n",
          ft.ft_month, ft.ft_day,
          ft.ft_year+1980);
   fclose(stream);
   return 0;
}
```

getgraphmode

Funktion — Ermittelt den aktuellen Grafikmodus.

Syntax

```
#include <graphics.h>
int far getgraphmode(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Vor der Verwendung von *getgraphmode* muß ein erfolgreicher Aufruf von *initgraph* durchgeführt worden sein.

Der Aufzählungstyp *graphics_mode* – definiert in graphics.h – enthält Namen für die vordefinierten Grafikmodi. Eine Tabelle hierzu finden Sie in der Beschreibung von *initgraph*.

Rückgabewert — *getgraphmode* liefert die Nummer des durch *initgraph* oder *setgraphmode* gesetzten Grafikmodus zurück.

Siehe auch — *getmoderange, restorecrtmode, setgraphmode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
```

```
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, mode;
   char numname[80], modename[80];

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

/* get mode number and name strings */
   mode = getgraphmode();
   sprintf(numname, "%d is the current mode number.",
           mode);
   sprintf(modename,"%s is the current graphics mode",
           getmodename(mode));

/* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, numname);
   outtextxy(midx, midy+2*textheight("W"), modename);

/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getimage

Funktion — Kopiert einen rechteckigen Bildausschnitt in den Speicher.

Syntax

```
#include <graphics.h>
void far getimage(int left, int top, int right,
                  int bottom, void far *bitmap);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getimage* kopiert einen Bildausschnitt, dessen obere linke Ecke durch (*left*, *top*) und dessen untere rechte Ecke durch (*right*, *bottom*) festgelegt ist, in die als *bitmap* übergebene Variable. Die ersten beiden 16-Bit-Worte von *bitmap* werden für die Breite und Höhe des kopierten Ausschnitts reserviert, der Bildausschnitt selbst wird im restlichen Teil von *bitmap* gespeichert.

Rückgabewert Keiner.

Siehe auch *imagesize*, *putimage*, *putpixel*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

void save_screen(void far *buf[4]);
void restore_screen(void far *buf[4]);

int maxx, maxy;

int main(void)
{
   int gdriver=DETECT, gmode, errorcode;
   void far *ptr[4];

   /* auto-detect the graphics driver and mode */
   initgraph(&gdriver, &gmode, "");
   errorcode = graphresult();
               /* check for any errors */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
   maxx = getmaxx();
   maxy = getmaxy();

   /* draw an image on the screen */
   rectangle(0, 0, maxx, maxy);
   line(0, 0, maxx, maxy);
   line(0, maxy, maxx, 0);
```

```
   save_screen(ptr);    /* save the current screen */
   getch();             /* pause screen */
   cleardevice();       /* clear screen */
   restore_screen(ptr); /* restore the screen */
   getch();             /* pause screen */

   closegraph();
   return 0;
}

void save_screen(void far *buf[4])
{
   unsigned size;
   int ystart=0, yend, yincr, block;

   yincr = (maxy+1) / 4;
   yend = yincr;
   size = imagesize(0, ystart, maxx, yend);
/* get byte size of image */

   for (block=0; block<=3; block++)
   {
      if ((buf[block] = farmalloc(size)) == NULL)
      {
         closegraph();
         printf("Error: not enough heap space in "
                "save_screen().\n");
         exit(1);
      }

      getimage(0, ystart, maxx, yend, buf[block]);
      ystart = yend + 1;
      yend += yincr + 1;
   }
}

void restore_screen(void far *buf[4])
{
   int ystart=0, yend, yincr, block;

   yincr = (maxy+1) / 4;
   yend = yincr;

   for (block=0; block<=3; block++)
   {
      putimage(0, ystart, buf[block], COPY_PUT);
      farfree(buf[block]);
```

```
      ystart = yend + 1;
      yend += yincr + 1;
   }
}
```

getlinesettings

Funktion Ermittelt die aktuelle Linienart, -muster und -dicke.

Syntax

```
#include <graphics.h>
void far getlinesettings(struct linesettingstype
                         far *lineinfo);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getlinesettings* ermittelt die aktuelle Linienart, das verwendete Linienmuster sowie die Liniendicke und speichert diese Informationen in der Strukur vom Typ *linesettingstype,* auf die *lineinfo* zeigt.

Der Strukturtyp *linesettingstype* ist in graphics.h folgendermaßen definiert:

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
};
```

linestyle gibt die Linienart an, in der im folgenden Linien gezeichnet werden (z.B. durchgezogen, gepunktet, gestrichelt, usw.). Der Aufzählungstyp *line_styles*, definiert in graphics.h, stellt Namen für diese Operatoren zur Verfügung:

Name	Wert	Beschreibung
SOLID_LINE	0	durchgezogen
DOTTED_LINE	1	gepunktet
CENTER_LINE	2	Punkt Strich Punkt Strich...
DASHED_LINE	3	gestrichelt
USERBIT_LINE	4	benutzerdefiniert

Für die Dicke einer Linie (*thickness*) sind nur zwei Werte möglich:

Name	Wert	Beschreibung
NORM_WIDTH	1	normal (1 Pixel)
THICK_WIDTH	3	dick (3 Pixel)

upattern ist ein 16-Bit-Muster und hat nur dann eine Wirkung, wenn *linestyle* den Wert USERBIT_LINE (4) hat. In diesem Fall wird ein Pixel in der aktuellen Zeichenfarbe gezeichnet, wenn das korrespondierende Bit auf 1 gesetzt ist. Zum Beispiel ergibt 0xFFFF für *upattern* eine durchgezogene Linie, 0x3333 eine gestrichelte Linie usw.

Auch dann, wenn *setlinestyle* den Wert von *uppattern* für *linestyle != USERBIT_LINE* ignoriert, muß dieser Parameter immer angegeben werden.

Rückgabewert Keiner.

Siehe auch *setlinestyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE",
                  "DOTTED_LINE",
                  "CENTER_LINE",
                  "DASHED_LINE",
                  "USERBIT_LINE"
                };

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct linesettingstype lineinfo;
   int midx, midy;
   char lstyle[80], lpattern[80], lwidth[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
```

```
    midy = getmaxy() / 2;

    /* get information about current line settings */
    getlinesettings(&lineinfo);

    /* convert line information into strings */
    sprintf(lstyle, "%s is the line style.",
            lname[lineinfo.linestyle]);
    sprintf(lpattern, "0x%X is the user-defined "
            "line pattern.", lineinfo.upattern);
    sprintf(lwidth, "%d is the line thickness.",
            lineinfo.thickness);

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, lstyle);
    outtextxy(midx, midy+2*textheight("W"), lpattern);
    outtextxy(midx, midy+4*textheight("W"), lwidth);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getmaxcolor

Funktion — Liefert den größten Farbwert zurück, der an die Funktion *setcolor* übergeben werden kann.

Syntax

```
#include <graphics.h>
int far getmaxcolor(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getmaxcolor* liefert den größten gültigen Farbwert, der für den aktuellen Grafikmodus und -treiber an *setcolor* übergeben werden kann.

Zum Beispiel liefert *getmaxcolor* auf einer EGA-Karte mit 256 KBytes immer 15 zurück, was bedeutet, daß jeder Aufruf von *setcolor* mit einem Wert zwischen 0 und 15 gültig ist. Auf einer CGA-Karte im hochauflösenden Modus oder auf einem Hercules-Monochromadapter liefert *getmaxcolor* den Wert 1.

Rückgabewert — *getmaxcolor* liefert den höchsten verfügbaren Farbwert zurück.

Siehe auch — *getbkcolor*, *getcolor*, *getpalette*, *getpalettesize*, *setcolor*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char colstr[80];

   /* initialize graphics and local variables */
     initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* * * * * * * * * * * * * * * * * * * * * * * * * */
   /*                                                 */
   /* grab the color info. and convert it to a        */
   /* string                                          */
   sprintf(colstr, "This mode supports colors 0..%d",
           getmaxcolor());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, colstr);
   /*                                                 */
   /* * * * * * * * * * * * * * * * * * * * * * * * * */

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getmaxmode

Funktion — Liefert die höchste Nummer der Grafikmodi des aktuellen Treibers.

Syntax

```
include <graphics.h>
int far getmaxmode(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getmaxmode* ermittelt den maximal möglichen Wert für *graphmode* durch einen direkten Aufruf des aktuellen Treibers. Diese Funktion ist im Gegensatz zu *getmoderange* auch für Grafiktreiber verwendbar, die nicht von Borland erstellt wurden. Der niedrigste Grafikmodus ist immer 0.

Rückgabewert — Zurückgeliefert wird der »höchste« Grafikmodus für den aktuellen Treiber.

Siehe auch — *getmodename, getmoderange*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
       /* request auto detection */
       int gdriver = DETECT, gmode, errorcode;
       int midx, midy;
       char modestr[80];

      /* initialize graphics and local variables */
      initgraph(&gdriver, &gmode, "");

      /* read result of initialization */
      errorcode = graphresult();
      if (errorcode != grOk)  /* an error occurred */
{
          printf("Graphics error: %s\n",
                 grapherrormsg(errorcode));
          printf("Press any key to halt:");
          getch();
          exit(1); /* terminate with an error code */
       }

       midx = getmaxx() / 2;
       midy = getmaxy() / 2;
  /* grab the mode info. and convert it to a string */
```

```
        sprintf(modestr, "This driver supports "
                "modes 0..%d", getmaxmode());

      /* display the information */
      settextjustify(CENTER_TEXT, CENTER_TEXT);
      outtextxy(midx, midy, modestr);

      /* clean up */
      getch();
      closegraph();
     return 0;
}
```

getmaxx

Funktion — Liefert die maximale X-Koordinate des Bildschirms.

Syntax

```
#include <graphics.h>
int far getmaxx(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getmaxx* liefert in Abhängigkeit vom aktuellen Grafiktreiber und -modus die maximal mögliche X-Koordinate des Bildschirms zurück.

Beispielsweise liefert *getmaxx* auf einem *Color Graphics Adapter* im Modus CGAC0 (320 * 200 Pixel) 319 zurück. *getmaxx* ist besonders geeignet zum Zentrieren oder Festlegen von Ausschnittgrenzen auf dem Bildschirm usw.

Rückgabewert — *getmaxx* liefert die maximal mögliche X-Koordinate des Bildschirms zurück.

Siehe auch — *getmaxy, getx*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char xrange[80], yrange[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
```

```
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* convert max resolution values into strings */
   sprintf(xrange, "X values range from 0..%d",
           getmaxx());
   sprintf(yrange, "Y values range from 0..%d",
           getmaxy());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, xrange);
   outtextxy(midx, midy+textheight("W"), yrange);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getmaxy

Funktion — Liefert die maximal mögliche Y-Koordinate des Bildschirms.

Syntax

```
#include <graphics.h>
int far getmaxy(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getmaxy* liefert in Abhängigkeit vom aktuellen Grafiktreiber und -modus die maximal mögliche Y-Koordinate des Bildschirms zurück.

Beispielsweise liefert *getmaxy* auf einem *Color Graphics Adapter* im Modus CGAC0 (320 * 200 Pixel) 199 zurück. *getmaxy* ist besonders geeignet zum Zentrieren oder Festlegen von Ausschnittgrenzen auf dem Bildschirm usw.

Rückgabewert — *getmaxy* liefert die maximal mögliche Y-Koordinate des Bildschirms zurück.

Siehe auch *getmaxx, getx, gety*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char xrange[80], yrange[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* convert max resolution values into strings */
   sprintf(xrange, "X values range from 0..%d",
           getmaxx());
   sprintf(yrange, "Y values range from 0..%d",
           getmaxy());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, xrange);
   outtextxy(midx, midy+textheight("W"), yrange);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getmodename

Funktion Liefert einen Zeiger auf einen String mit dem Namen des angegebenen Grafikmodus.

Syntax

```
#include <graphics.h>
char *far getmodename(int mode_number);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getmodename* erwartet die Nummer eines Grafikmodus als Parameter und liefert einen Zeiger auf den Namen dieses Modus zurück. Die Namen der Grafikmodi sind Bestandteil des jeweiligen Treibers. Die Rückgabewerte ("320x200 CGA P1", "640x200 CGA", usw.) sind nützlich, um Menüs aufzubauen oder den Status anzuzeigen.

Rückgabewert *getmodename* liefert einen Zeiger auf einen String mit dem Namen des Grafikmodus.

Siehe auch *getmaxmode, getmoderange*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, mode;
   char numname[80], modename[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get mode number and name strings */
   mode = getgraphmode();
   sprintf(numname, "%d is the current mode number.",
           mode);
```

```
   sprintf(modename, "%s is the current graphics "
           "mode.", getmodename(mode));

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, numname);
   outtextxy(midx, midy+2*textheight("W"), modename);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getmoderange

Funktion

Ermittelt den Bereich möglicher Grafikmodi des angegebenen Grafiktreibers.

Syntax

```
#include <graphics.h>
void far getmoderange(int graphdriver,
                      int far *lomode,
                      int far *himode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

getmoderange liefert den niedrigstmöglichen Grafikmodus für den Grafiktreiber *graphdriver* in **lomode* und den höchstmöglichen Modus in **himode* zurück. Wenn *graphdriver* den Wert eines ungültigen Grafiktreibers enthält, werden **lomode* und **himode* auf -1 gesetzt. Wenn *graphdriver* den Wert -1 enthält, werden die Modi des aktuell geladenen Treibers übergeben.

Rückgabewert

Keiner.

Siehe auch

getgraphmode, getmaxmode, getmodename, initgraph, setgraphmode

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int low, high;
   char mrange[80];
```

```
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get the mode range for this driver */
   getmoderange(gdriver, &low, &high);

   /* convert mode range info. into strings */
   sprintf(mrange,"This driver supports modes %d..%d",
           low, high);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, mrange);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getpalette

Funktion Liefert Informationen über die aktuelle Farbpalette.

Syntax
```
#include <graphics.h>
void far getpalette(struct palettetype far *palette);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *getpalette* liefert die Farben der aktuellen Farbpalette und ihre Gesamtgröße in der Struktur vom Typ *palettetype* zurück, auf die *palette* zeigt.

Die Konstante MAXCOLORS und die Struktur *palettetype* sind in graphics.h folgendermaßen definiert:

```
#define MAXCOLORS 15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS+1];
};
```

Das Element *size* enthält die Anzahl der tatsächlich benutzten Einträge des Array *colors*, sein Wert ist vom aktiven Grafikmodus und dem verwendeten Treiber abhängig.

Die Elemente von *colors* (ein Array mit *size* Bytes) enthalten die aktuellen Farbnummern für jeden Eintrag in der Palette.

Hinweis: getpalette kann nicht zusammen mit dem IBM-8514-Adapter verwendet werden.

Rückgabewert Keiner.

Siehe auch *getbkcolor, getcolor, getdefaultpalette, getmaxcolor, setallpalette, setpalette*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct palettetype pal;
   char psize[80], pval[20];
   int i, ht;
   int y = 10;

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
   }
```

```
/* grab a copy of the palette */
   getpalette(&pal);

/* convert palette info. into strings */
   sprintf(psize, "The palette has %d "
           "modifiable entries.", pal.size);

/* display the information */
   outtextxy(0, y, psize);
   if (pal.size != 0)
   {
      ht = textheight("W");
      y += 2*ht;
      outtextxy(0, y, "Here are the current values:");
      y += 2*ht;
      for (i=0; i<pal.size; i++, y+=ht)
      {
         sprintf(pval, "palette[%02d]: 0x%02X",
                 i, pal.colors[i]);
         outtextxy(0, y, pval);
      }
   }

/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getpalettesize

Funktion — Liefert die Größe der aktuellen Farbpalette zurück.

Syntax

```
#include <graphics.h>
int far getpalettesize(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Mit *getpalettesize* läßt sich ermitteln, wieviele Einträge in der Farbpalette im aktuellen Grafikmodus gesetzt werden können. Beispielsweise wird für eine EGA-Karte im Farbmodus 16 zurückgeliefert.

Rückgabewert — *getpalettesize* liefert die Anzahl der Einträge in der aktuellen Farbpalette zurück.

Siehe auch — *setpalette, setallpalette*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>

int main()
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char psize[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* convert palette size info. into string */
   sprintf(psize, "The palette has %d "
          "modifiable entries.", getpalettesize());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, psize);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getpass

Funktion Liest ein Paßwort.

Syntax

```
#include <conio.h>
char *getpass(const char *prompt);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x			

Beschreibung *getpass* gibt den durch *prompt* bezeichneten nullterminierten String aus und erwartet eine Eingabe des Benutzers über die Tastatur, die mit Return abgeschlossen werden muß. Die vom Benutzer eingegebenen Zeichen werden *nicht* auf dem Bildschirm dargestellt.

Der String hat eine Maximallänge von acht Zeichen (wobei das terminierende Nullzeichen nicht mitgezählt ist).

Rückgabewert *getpass* liefert einen Zeiger auf einen statischen String zurück, der durch jeden erneuten Aufruf überschrieben wird.

Siehe auch *getch*

Beispiel

```
#include <conio.h>

int main(void)
{
   char *password;

   password = getpass("Input a password:");
   cprintf("The password is: %s\r\n", password);
   return 0;
}
```

getpid

Funktion Liefert die Prozeß-ID eines Programms.

Syntax

```
#include <process.h>
unsigned getpid(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung Die Prozeß-ID dient zur eindeutigen Kennzeichnung eines Programms. Dieses Konzept stammt aus Multitasking-Betriebssystemen wie UNIX, in denen jeder Prozeß eine eindeutige Prozeßnummer hat.

Rückgabewert *getpid* liefert den Segmentwert des PSP eines Programms zurück.

Siehe auch *getpsp*, *_psp* (globale Variable)

Beispiel

```
#include <stdio.h>
#include <process.h>

int main()
{
   printf("This program's process identification "
          "number (PID) number is %    X\n", getpid());
```

```
   printf("Note: under DOS it is the PSP segment\n");
   return 0;
}
```

getpixel

Funktion — Ermittelt die Farbe eines Pixels.

Syntax

```
#include <graphics.h>
unsigned far getpixel(int x, int y);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getpixel* liefert die Farbe des Pixels auf dem durch *(x,y)* angegebenen Punkt zurück.

Rückgabewert — *getpixel* liefert die Farbe des angegebenen Pixels zurück.

Siehe auch — *getimage, putpixel*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME  100  /* in milliseconds */

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int i, x, y, color, maxx, maxy,
       maxcolor, seed;

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
/* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
/* terminate with an error code */
      exit(1);
```

```
   }
   maxx = getmaxx() + 1;
   maxy = getmaxy() + 1;
   maxcolor = getmaxcolor() + 1;

   while (!kbhit())
   {
/* seed the random number generator */
      seed = random(32767);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++)
      {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         putpixel(x, y, color);
      }

      delay(DELAY_TIME);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++)
      {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         if (color == getpixel(x, y))
            putpixel(x, y, 0);
      }
   }

/* clean up */
   getch();
   closegraph();
   return 0;
}
```

getpsp

Funktion — Liefert die Adresse des Programmsegment-Präfix (PSP).

Syntax

```
#include <dos.h>
unsigned getpsp(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung — *getpsp* verwendet den DOS-Funktionsaufruf 0x62, um die Segmentadresse des Programmsegment-Präfix (PSP) zu ermitteln.

Rückgabewert — *getpsp* liefert die Segmentadresse des PSP zurück.

Siehe auch *getenv*, *_psp* (globale Variable),

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   static char command[128];
   char far *cp;
   int len, i;

   printf("The program segment prefix is: %u\n",
          getpsp());

/*
  _psp is preset to segment of the PSP
   Command line is located at offset 0x81
   from start of PSP
*/
   cp = (char *) MK_FP(_psp, 0x80);
   len = *cp;

   for (i = 0; i < len; i++)
      command[i] = cp[i+1];

   printf("Command line: %s\n", command);

   return 0;
}
```

gets

Funktion Liest einen String aus *stdin*.

Syntax

```
#include <stdio.h>
char *gets(char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung *gets* liest eine Zeichenfolge aus dem Standardeingabe-Stream *stdin* und speichert sie in der durch *s* angegebenen Stringvariablen. Die Eingabe wird durch einen Zeilenvorschub beendet, der in *s* als Nullzeichen (\0) gespeichert wird.

Eingabestrings dürfen bestimmte Whitespace-Zeichen (Leerzeichen, Tabs) enthalten. *gets* kehrt zurück, wenn ein Zeilenvorschub gelesen wird; die Zeichenfolge bis zu diesem Zeilenvorschub wird in *s* gespeichert.

Rückgabewert *gets* liefert bei fehlerfreier Ausführung die Stringvariable *s* zurück. Im Fall eines Fehlers oder wenn beim Lesen das Dateiende überschritten wird, ist der Rückgabewert NULL.

Siehe auch *cgets, ferror, fgets, fopen, fputs, fread, getc, puts, scanf*

Beispiel

```
#include <stdio.h>

int main(void)
{
   char string[80];

   printf("Input a string:");
   gets(string);
   printf("The string input was: %s\n", string);
   return 0;
}
```

gettext

Funktion Kopiert Text vom Textbildschirm in den Speicher.

Syntax

```
#include <conio.h>
int gettext(int left, int top, int right, int bottom,
            void *destin);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *gettext* speichert den Inhalt eines rechteckigen Textbildschirmausschnitts in dem Speicherbereich, auf den *destin* zeigt. Die linke obere Ecke des Ausschnitts ist durch (*left*,*top*), die rechte untere Ecke durch (*bottom*,*right*) festgelegt.

Die Funktion arbeitet unabhängig von einem eventuell gesetzten Textfenster mit absoluten Bildschirmkoordinaten, wobei die obere linke Ecke die Koordinaten (1,1) hat.

Der Inhalt des Rechtecks wird von *gettext* sequentiell von links nach rechts und von oben nach unten in den Speicher eingelesen.

Jede Bildschirmposition benötigt zwei Bytes Speicherplatz: Das erste Byte ist das Zeichen, das zweite das Video-Attribut an der jeweiligen Position. Die benötigte Größe für ein Rechteck mit *w* Spalten und *h* Zeilen ergibt sich aus:

Bytes = (*h* Zeilen) * (*w* Spalten) * 2

Rückgabewert Bei fehlerfreier Ausführung liefert *gettext* den Wert 1, im Fall eines Fehlers (wenn Sie z.B. Koordinaten außerhalb des aktuellen Bildschirmmodus angegeben haben) 0 zurück.

Siehe auch *movetext, puttext*

Beispiel

```
#include <conio.h>

char buffer[4096];
```

```
int main(void)
{
   int i;

   clrscr();
   for (i = 0; i <= 20; i++)
      cprintf("Line #%d\r\n", i);
   gettext(1, 1, 80, 25, buffer);

   gotoxy(1, 25);
   cprintf("Press any key to clear screen...");
   getch();
   clrscr();
   gotoxy(1, 25);
   cprintf("Press any key to restore screen...");
   getch();
   puttext(1, 1, 80, 25, buffer);
   gotoxy(1, 25);
   cprintf("Press any key to quit...");
   getch();

   return 0;
}
```

gettextinfo

Funktion Ermittelt Informationen über den aktuellen Textmodus.

Syntax
```
#include <conio.h>
void gettextinfo(struct text_info *r);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *gettextinfo* setzt die Elemente der über *r* übergebenen Struktur des Typs *text_info* mit den aktuellen Parametern für Textausgaben. Der Strukturtyp *text_info* ist in conio.h folgendermaßen definiert:

```
struct text_info {
  unsigned char winleft;
      /* linke Fensterkoordinate */
  unsigned char wintop;
      /* obere Fensterkoordinate */
  unsigned char winright;......
      /* rechte Fensterkoordinate */
  unsigned char winbottom;.....
      /* untere Fensterkoordinate */
  unsigned char attribute;
      /* Textattribut */
```

```
  unsigned char normattr;
      /* normales Attribut */
  unsigned char currmode;
      /* Videomodus: BW40, BW80, C40, C80 oder C4350 */
  unsigned char screenheight;
      /* Höhe des Textfensters */
  unsigned char screenwidth;
      /* Breite */
  unsigned char curx,
      /* Cursorposition */
  unsigned char cury;
      /* innerhalb des aktuellen Textfensters */
};
```

Rückgabewert Keiner. Die Ergebnisse werden in der Struktur, auf die *r* zeigt, zurückgeliefert.

Siehe auch *textattr, textbackground, textcolor, textmode, wherex, wherey, window*

Beispiel

```
#include <conio.h>

int main(void)
{
   struct text_info ti;
   gettextinfo(&ti);
   cprintf("window left      %2d\r\n",ti.winleft);
   cprintf("window top       %2d\r\n",ti.wintop);
   cprintf("window right     %2d\r\n",ti.winright);
   cprintf("window bottom    %2d\r\n",ti.winbottom);
   cprintf("attribute        %2d\r\n",ti.attribute);
   cprintf("normal attribute %2d\r\n",ti.normattr);
   cprintf("current mode     %2d\r\n",ti.currmode);
   cprintf("screen height    %2d\r\n",
           ti.screenheight);
   cprintf("screen width     %2d\r\n",ti.screenwidth);
   cprintf("current x        %2d\r\n",ti.curx);
   cprintf("current y        %2d\r\n",ti.cury);
   return 0;
}
```

gettextsettings

Funktion Ermittelt Informationen über den aktuellen Grafik-Font.

Syntax

```
#include <graphics.h>
void far gettextsettings(struct textsettingstype
                         far *texttypeinfo);
```

DOS	UNIX	Windows	ANSI C	nur C++
	x			

Beschreibung *gettextsettings* ermittelt die aktuell gesetzten Parameter über den aktuellen Font, die Richtung (horizontal oder vertikal), den Vergrößerungsfaktor und die Ausrichtung und liefert sie in der über *texttypeinfo* übergebenen Struktur vom Typ *textsettingstype* zurück.

Der Strukturtyp *textsettingstype* ist in graphics.h folgendermaßen definiert:

```
struct textsettingstype {
  int font;      /* Nummer des aktiven Zeichensatzes */
  int direction; /* horizontal oder vertikal */
  int charsize;  /* Vergrößerungsfaktor */
  int horiz;     /* horizontale Ausrichtung */
  int vert;      /* vertikale Ausrichtung */
};
```

Eine Beschreibung der Felder finden Sie in der Beschreibung von *settextstyle*.

Rückgabewert Keiner.

Siehe auch *outtext, outtextxy, registerbgifont, settextjustify, settextstyle, setusercharsize, textheight, textwidth*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fonts supported */
char *font[] = { "DEFAULT_FONT",
                 "TRIPLEX_FONT",
                 "SMALL_FONT",
                 "SANS_SERIF_FONT",
                 "GOTHIC_FONT"
               };

/* the names of the text directions supported */
char *dir[] = { "HORIZ_DIR", "VERT_DIR" };

/* horizontal text justifications supported */
char *hjust[] = { "LEFT_TEXT", "CENTER_TEXT",
                  "RIGHT_TEXT" };

/* vertical text justifications supported */
char *vjust[] = { "BOTTOM_TEXT", "CENTER_TEXT",
                  "TOP_TEXT" };
```

```
int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct textsettingstype textinfo;
   int midx, midy, ht;
   char fontstr[80], dirstr[80], sizestr[80];
   char hjuststr[80], vjuststr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get information about current text settings */
   gettextsettings(&textinfo);

   /* convert text information into strings */
   sprintf(fontstr, "%s is the text style.",
           font[textinfo.font]);
   sprintf(dirstr, "%s is the text direction.",
           dir[textinfo.direction]);
   sprintf(sizestr, "%d is the text size.",
           textinfo.charsize);
   sprintf(hjuststr, "%s is the horizontal "
           "justification.", hjust[textinfo.horiz]);
   sprintf(vjuststr, "%s is the vertical "
           "justification.", vjust[textinfo.vert]);

   /* display the information */
   ht = textheight("W");
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, fontstr);
   outtextxy(midx, midy+2*ht, dirstr);
   outtextxy(midx, midy+4*ht, sizestr);
   outtextxy(midx, midy+6*ht, hjuststr);
```

```
   outtextxy(midx, midy+8*ht, vjuststr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

gettime, settime

Funktion — Lesen und Setzen der Uhrzeit des Systems.

Syntax

```
#include <dos.h>
void gettime(struct time *timep);
void settime(struct time *timep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *gettime* liefert in der über *timep* angegebenen Struktur des Typs *time* die aktuelle Uhrzeit des Systems zurück.

settime setzt die Uhrzeit des Systems auf die Werte, die in der über *timep* angegebenen Struktur *time* enthalten sind.

Der Strukturtyp *time* ist in dos.h folgendermaßen definiert:

```
struct time {
   unsigned char ti_min;     /* Minuten */
   unsigned char ti_hour;    /* Stunden */
   unsigned char ti_hund;    /* 1/100 Sekunden */
   unsigned char ti_sec;     /* Sekunden */
};
```

Rückgabewert — Keiner.

Siehe auch — *_dos_gettime, _dos_settime, getdate, setdate, stime, time*

Beispiel

```
#include   <stdio.h>
#include   <dos.h>

int main(void)
{
   struct  time t;

   gettime(&t);
   printf("The current time is: %2d:%02d:%02d.%02d\n",
          t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
   return 0;
}
```

getvect, setvect

Funktion — Lesen und Setzen eines Interrupt-Vektors.

Syntax

```
#include <dos.h>
void interrupt(*getvect(int interruptno))();
void setvect(int interruptno, void interrupt (*isr)());
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — Die Prozessoren der 8086-Familie reservieren eine Reihe von Interrupt-Vektoren, die von 0 bis 255 durchnumeriert sind. Der vier Byte lange Wert jedes Vektors ist die Adresse einer Interrupt-Funktion.

getvect liefert den Wert des Vektors *interruptno* als **far**-Zeiger auf eine Funktion des Typs **interrupt** zurück. Der Wert von *interruptno* muß zwischen 0 und 255 liegen.

setvect setzt den Vektor mit der Nummer *interruptno* so, daß er auf die als *isr* (interrupt service routine) angegebene C-Funktion zeigt. Diese C-Funktion muß als **interrupt** -Routine deklariert worden sein.

Hinweis: Solange die in dos.h deklarierten Prototypen verwendet werden, ist *setvect* ohne weitere Modifikationen durch Übergabe der Adresse in jedem Speichermodell anwendbar.

Rückgabewert — *getvect* liefert den aktuellen vier Byte langen Wert zurück, der im Interrupt-Vektor *interruptno* gespeichert ist.

setvect liefert keinen Wert zurück.

Siehe auch — *disable*, *_dos_getvect*, *_dos_setvect*, *enable*, *geninterrupt*

Beispiel

```
#include <stdio.h>
#include <dos.h>

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt get_out(__CPPARGS);
                /* interrupt prototype */
void interrupt (*oldfunc)(__CPPARGS);
                /* interrupt function pointer */

int looping = 1;

int main(void)
{
  puts("Press <Shift><Prt Sc> to terminate");
```

```
  /* save the old interrupt */
  oldfunc  = getvect(5);

   /* install interrupt handler */
  setvect(5,get_out);

   /* do nothing */
  while (looping);

   /* restore to original interrupt routine */
   setvect(5,oldfunc);

  puts("Success");
  return 0;
}

void interrupt get_out(__CPPARGS)
{
looping = 0;
    /* change global variable to get out of loop */
}
```

getverify

Funktion — Liefert den Status des DOS-Flag *Verify*.

Syntax

```
#include <dos.h>
int getverify(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *getverify* liefert den aktuellen Status des Flag *Verify* zurück.

Über das Flag *Verify* wird festgelegt, ob Schreiboperationen auf der Diskette automatisch überprüft werden oder nicht.

Wenn dieses Flag gesetzt ist, werden geschriebene Daten auf ihre Korrektheit überprüft.

Rückgabewert — *getverify* liefert den aktuellen Status des *Verify*-Flags

0 = Überprüfung aus
1 = Überprüfung an

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   if (getverify())
```

```
      printf("DOS verify flag is on\n");
   else
      printf("DOS verify flag is off\n");
   return 0;
}
```

getviewsettings

Funktion — Ermittelt Informationen über das aktuelle Zeichenfenster.

Syntax

```
#include <graphics.h>
void far getviewsettings(struct viewporttype
                         far *viewport);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getviewsettings* speichert in den Elementen der Struktur vom Typ *viewporttype*, auf die *viewport* zeigt, Informationen über das aktuelle Zeichenfenster.

Der Strukturtyp *viewporttype* ist in graphics.h folgendermaßen definiert:

```
struct viewporttype {
   int left, top,        /* obere linke und     */
       right, bottom;    /* untere rechte Ecke  */
   int clip;             /* Clip-Funktion an/aus */
};
```

Rückgabewert — Keiner.

Siehe auch — *clearviewport, getx, gety, setviewport*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

char *clip[] = { "OFF", "ON" };

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct viewporttype viewinfo;
   int midx, midy, ht;
   char topstr[80], botstr[80], clipstr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
   /* read result of initialization */
```

```
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get information about current viewport */
   getviewsettings(&viewinfo);

   /* convert text information into strings */
   sprintf(topstr, "(%d, %d) is the upper left "
           "viewport corner.", viewinfo.left,
           viewinfo.top);
   sprintf(botstr, "(%d, %d) is the lower right "
           "viewport corner.", viewinfo.right,
           viewinfo.bottom);
   sprintf(clipstr, "Clipping is turned %s.",
           clip[viewinfo.clip]);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   ht = textheight("W");
   outtextxy(midx, midy, topstr);
   outtextxy(midx, midy+2*ht, botstr);
   outtextxy(midx, midy+4*ht, clipstr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

getw

Funktion: Liest einen Integerwert aus einem Stream.

Syntax:

```
#include <stdio.h>
int getw(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *getw* liest den nächsten Integer aus dem durch *stream* bezeichneten Eingabe-Stream. Eine spezielle Anordnung des Dateiinhalts wird nicht vorausgesetzt.

getw sollte nicht benutzt werden, wenn der Stream im Textmodus geöffnet wurde.

Rückgabewert *getw* liest zwei Byte aus der angegebenen Datei und liefert sie als **int** zurück. Bei Überschreitung des Dateiendes ist der Rückgabewert EOF. Da EOF aber auch einen legitimen Wert darstellen kann, sollte ein expliziter Test mit *feof* oder *ferror* folgen, um auf Dateiende oder Fehler zu prüfen.

Siehe auch *putw*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>

#define FNAME "test.$$$"

int main(void)
{
   FILE *fp;
   int word;

   /* place the word in a file */
   fp = fopen(FNAME, "wb");
   if (fp == NULL)
   {
      printf("Error opening file %s\n", FNAME);
      exit(1);
   }

   word = 94;
   putw(word,fp);
   if (ferror(fp))
       printf("Error writing to file\n");
   else
       printf("Successful write\n");
   fclose(fp);

   /* reopen the file */
   fp = fopen(FNAME, "rb");
   if (fp == NULL)
   {
      printf("Error opening file %s\n", FNAME);
      exit(1);
   }

   /* extract the word */
   word = getw(fp);

   if (ferror(fp))
       printf("Error reading file\n");
```

```
    else
        printf("Successful read: word = %d\n", word);

    /* clean up */
    fclose(fp);
    unlink(FNAME);

    return 0;
}
```

getx

Funktion — Liefert die X-Koordinate der aktuellen Position (CP) zurück.

Syntax

```
#include <graphics.h>
int far getx(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *getx* liefert die X-Koordinate der aktuellen Grafikposition (CP).
Die von *getx* zurückgelieferten Koordinaten sind relativ zum aktuellen Zeichenfenster.

Rückgabewert — *getx* liefert die X-Koordinate der aktuellen Position (CP).

Siehe auch — *getmaxx, getmaxy, getviewsettings, gety, moveto*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
```

```
   /* move to the screen center point */
   moveto(getmaxx() / 2, getmaxy() / 2);

   /* create a message string */
   sprintf(msg, "<-(%d, %d) is here.",getx(),gety());

   /* display the message */
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

gety

Funktion Liefert die Y-Koordinate der aktuellen Position (CP) zurück.

Syntax
```
#include <graphics.h>
int far gety(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *gety* liefert die Y-Koordinate der aktuellen Grafikposition (CP).
Die von *gety* zurückgelieferten Koordinaten sind relativ zum aktuellen Zeichenfenster.

Rückgabewert *gety* liefert die Y-Koordinate der aktuellen Position (CP) zurück.

Siehe auch *getmaxx, getmaxy, getviewsettings, getx, moveto*

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %     s\n",
```

```
            grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* move to the screen center point */
   moveto(getmaxx() / 2, getmaxy() / 2);

   /* create a message string */
   sprintf(msg, "<-(%d, %d) is the here.",
           getx(), gety());

   /* display the message */
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

gmtime

Funktion

Rechnet Datum und Uhrzeit in die Greenwich Mean Time (GMT) um.

Syntax

```
#include <time.h>
struct tm *gmtime(const time_t *timer);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung

gmtime erwartet die Adresse eines Wertes, der von *time* zurückgeliefert wird und liefert einen Zeiger auf eine Struktur vom Typ *tm*, in der die einzelnen Komponenten der Uhrzeit gespeichert sind.

Die globale **long**-Variable *timezone* sollte den Unterschied zwischen der lokalen Zeit und der GMT in Sekunden enthalten (für MEZ ist *timezone = -1 * 60 * 60*). Die globale Variable *daylight* darf nur dann auf einen Wert ungleich Null gesetzt werden, wenn die Standardkonvertierung der USA für Sommer- und Winterzeit angewendet werden soll.

Der Strukturtyp *tm* ist in time.h folgendermaßen definiert:

```
struct tm {
   int tm_sec;    /* Sekunden */
   int tm_min;    /* Minuten */
   int tm_hour;   /* Stunde (0 bis 23) */
   int tm_mday;   /* Tag im Monat (1 bis 31) */
```

```
   int tm_mon;    /* Monat (0 bis 11) */
   int tm_year;   /* Jahr (Kalenderjahr minus 1900) */
   int tm_wday;   /* Wochentag (0 bis 6, Sonntag = 0)
*/
   int tm_yday;   /* Tag im Jahr (0 bis 365) */
   int tm_isdst;  /* Ungleich Null bei */
                  /* US-Sommerzeitkonvertierung */
};
```

Die Tageszeit wird in 24 Stunden gemessen, der Tag im Monat von 1 bis 31, der Monat von 0 bis 11 und der Tag im Jahr von 0 bis 365. Der Wochentag beginnt mit Sonntag = 0 und zur Jahresangabe muß 1900 addiert werden. Das Flag *tm_isdst* gibt an, ob die Sommerzeitkonvertierung durchgeführt wurde.

Rückgabewert *gmtime* liefert einen Zeiger auf eine statische Struktur mit der ermittelten Information, die durch jeden Aufruf dieser Funktion überschrieben wird.

Siehe auch *asctime*, *ctime*, *ftime*, *localtime*, *stime*, *time*, *tzset*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>

/* Pacific Standard Time & Daylight Savings */
char *tzstr = "TZ=PST8PDT";

int main(void)
{
   time_t t;
   struct tm *gmt, *area;

   putenv(tzstr);
   tzset();

   t = time(NULL);
   area = localtime(&t);
   printf("Local time is: %s", asctime(area));
   gmt = gmtime(&t);
   printf("GMT is:        %s", asctime(gmt));
   return 0;
}
```

gotoxy

Funktion Positioniert den Cursor im Textfenster.

Syntax

```
#include <conio.h>
void gotoxy(int x, int y);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *gotoxy* positioniert den Cursor auf die angegebene Position im aktuellen Textfenster.

Wenn die Koordinaten in irgendeiner Weise ungültig sind, wird der Aufruf von *gotoxy* vollständig ignoriert. Dies ist z.B.der Fall, wenn *gotoxy*(40,30) angegeben wird, die Koordinaten der rechten unteren Fensterecke aber bei (35,25) liegen.

Rückgabewert Keiner.

Siehe auch *wherex*, *wherey*, *window*

Beispiel

```
#include <conio.h>

int main(void)
{
   clrscr();
   gotoxy(35, 12);
   cprintf("Hello world");
   getch();
   return 0;
}
```

graphdefaults

Funktion Setzt sämtliche Einstellungen des Grafikpakets auf die Voreinstellungen zurück.

Syntax

```
#include <graphics.h>
void far graphdefaults(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *graphdefaults* setzt alle Einstellungen des Grafikpakets auf die Voreinstellungen:

- setzt das Zeichenfenster auf die vollständige Bildschirmgröße;
- positioniert die aktuelle Position (CP) auf (0,0);
- setzt die voreingestellte Hintergrundfarbe, Zeichenfarbe und Farbpalette;
- setzt die voreingestellte Art und das voreingestellte Muster für Flächenfüllungen;
- setzt den voreingestellten Font und die voreingestellte Ausrichtung.

Rückgabewert Keiner.

Siehe auch *initgraph, setgraphmode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* initialize graphics and local variables */
   initgraph(&gdriver,&gmode, "c:\\bor\\turbo5\\bgi");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();

   /* output line with non-default settings */
   setlinestyle(DOTTED_LINE, 0, 3);
   line(0, 0, maxx, maxy);
   outtextxy(maxx/2, maxy/3,
             "Before default values are restored.");
   getch();

   /* restore default values for everything */
   graphdefaults();

   /* clear the screen */
   cleardevice();

   /* output line with default settings */
   line(0, 0, maxx, maxy);
   outtextxy(maxx/2, maxy/3,
             "After restoring default values.");
```

```
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

grapherrormsg

Funktion — Liefert einen Zeiger auf eine Fehlermeldung des Grafikpakets.

Syntax

```
#include <graphics.h>
char *far grapherrormsg(int errorcode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *grapherrormsg* liefert einen Zeiger auf einen String zurück, der den Text der zu *errorcode* gehörigen Fehlermeldung enthält. Für *errorcode* wird der von *graphresult* gelieferte Wert benutzt.

In Kapitel 3 »Globale Variablen« finden Sie unter *errno* eine Liste aller möglichen Fehler und ihrer mnemonischen Bezeichnungen.

Rückgabewert — *grapherrormsg* liefert einen Zeiger auf einen String mit dem Text einer Fehlermeldung.

Siehe auch — *graphresult*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define NONSENSE -50

int main(void)
{
   /* FORCE AN ERROR TO OCCUR */
   int gdriver = NONSENSE, gmode, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   /* if an error occurred, then output a */
   /* descriptive error message.          */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
```

```
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

_graphfreemem

Funktion — Vom Grafikpaket verwendete Routine zur Freigabe von dynamisch belegtem Speicherplatz.

Syntax

```
#include <graphics.h>
void far _graphfreemem(void far *ptr, unsigned size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Die Grafikbibliothek ruft *_graphfreemem* auf, um einen vorher durch *_graphgetmem* belegten Bereich wieder freizugeben.

Es ist möglich, *_graphfreemem* durch eine eigene Version zu ersetzen: damit hat man die Kontrolle über die Speicherverwaltung der Grafik. Die Standardversion dieser Routine ruft nur *free* auf.

Die Deklaration Ihrer Version von *_graphfreemem* muß exakt in der unter »Syntax« gezeigten Form geschehen.

Rückgabewert — Keiner.

Siehe auch — *_graphgetmem, setgraphbufsize*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

int main(void)
{
       /* request auto detection */
       int gdriver = DETECT, gmode, errorcode;
       int midx, midy;

       /* clear the text screen */
```

```
        clrscr();
        printf("Press any key to initialize graphics "
               "mode:");
        getch();
        clrscr();

        /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

        /* read result of initialization */
        errorcode = graphresult();
        if (errorcode != grOk)  /* an error occurred */
        {
           printf("Graphics error: %s\n",
                  grapherrormsg(errorcode));
           printf("Press any key to halt:");
           getch();
           exit(1); /* terminate with an error code */
        }

        midx = getmaxx() / 2;
        midy = getmaxy() / 2;

        /* display a message */
        settextjustify(CENTER_TEXT, CENTER_TEXT);
        outtextxy(midx, midy, "Press any key to exit "
                  "graphics mode:");

        /* clean up */
        getch();
        closegraph();
        return 0;
}

/* called by the graphics kernel to allocate memory */
void far * far _graphgetmem(unsigned size)
{
        printf("_graphgetmem called to allocate "
               "%d bytes.\n", size);
        printf("press any key:");
        getch();
        printf("\n");

        /* allocate memory from far heap */
        return farmalloc(size);
}

/* called by the graphics kernel to free memory */
void far _graphfreemem(void far *ptr, unsigned size)
{
        printf("_graphfreemem called to free "
               %d bytes.\n", size);
```

```
        printf("press any key:");
        getch();
        printf("\n");

       /* free ptr from far heap */
       farfree(ptr);
}
```

_graphgetmem

Funktion

Vom Grafikpaket verwendete Routine zur dynamischen Belegung von Speicherplatz.

Syntax

```
#include <graphics.h>
void far *far _graphgetmem(unsigned size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

Die Routinen der Grafikbibliothek (nicht das Benutzerprogramm) rufen *_graphgetmem* auf, um Speicher für interne Puffer, Grafiktreiber und Schriften zu belegen.

Es ist möglich, *_graphgetmem* durch eine eigene Version zu ersetzen: damit hat man die Kontrolle über die Speicherverwaltung der Grafik. Die Standardversion dieser Routine ruft lediglich *malloc* auf.

Die Deklaration Ihrer Version von *_graphgetmem* muß exakt in der unter »Syntax« gezeigten Form geschehen.

Rückgabewert

Keiner.

Siehe auch

_graphfreemem, initgraph, setgraphbufsize

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* clear the text screen */
   clrscr();
   printf("Press any key to initialize "
          "graphics mode:");
```

```
   getch();
   clrscr();

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)      /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);    /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* display a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy,
             "Press any key to exit graphics mode:");

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* called by the graphics kernel to allocate memory */
void far * far _graphgetmem(unsigned size)
{
   printf("_graphgetmem called to allocate "
          "%d bytes.\n", size);
   printf("press any key:");
   getch();
   printf("\n");

   /* allocate memory from far heap */
   return farmalloc(size);
}

/* called by the graphics kernel to free memory */
void far _graphfreemem(void far *ptr, unsigned size)
{
   printf("_graphfreemem called to free %d bytes.\n",
          size);
   printf("press any key:");
   getch();
   printf("\n");
```

```
    /* free ptr from far heap */
    farfree(ptr);
}
```

graphresult

Funktion

Liefert den Fehlercode der letzten fehlgeschlagenen Grafikoperation zurück.

Syntax

```
#include <graphics.h>
int far graphresult(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

graphresult liefert den Fehlercode der letzten Grafikoperation, die einen Fehler erzeugt hat, und setzt den Fehlerstatus auf *grOk* zurück.

Die folgende Tabelle enthält eine Auflistung der Fehlernummern und Meldungen, die von *graphresult* zurückgeliefert werden. Der Aufzählungstyp *graph_errors*, der in graphics.h deklariert ist, definiert die Werte in dieser Tabelle.

Name	Wert	Text/Beschreibung
grOk	0	*No error* Fehlerfreie Ausführung
grNoInitGraph	-1	(*BGI*) *graphics not installed* (*use initgraph*) Grafiktreiber (.BGI) nicht geladen – *initgraph* muß zuerst aufgerufen werden
grNotDetected	-2	*Graphics hardware not detected* Kein grafikfähiger Adapter gefunden
grFileNotFound	-3	*Device driver file not found* Grafiktreiberdatei nicht gefunden
grInvalidDriver	-4	*Invalid device driver file* Grafiktreiberdatei defekt
grNoLoadMem	-5	*Not enough memory to load driver* Nicht genug Platz im Hauptspeicher für den Grafiktreiber
grNoScanMem	-6	*Out of memory in scan fill* Nicht genug Platz im Hauptspeicher (*bar*, *fillpoly*)
grNoFloodMem	-7	*Out of memory in flood fill* Nicht genug Platz im Hauptspeicher (*floodfill*)
grFontNotFound	-8	*Font file not found* Font-Datei nicht gefunden

Name	Wert	Text/Beschreibung
grNoFontMem	-9	*Not enough memory to load font* Nicht genug Platz im Hauptspeicher für den Font
grInvalidMode	-10	*Invalid Graphics mode for selected driver* Grafikmodus wird vom gewählten Treiber nicht unterstützt
grError	-11	*Graphics Error* Nicht näher bestimmbarer Fehler
grIOerror	-12	*Graphics I/O error* Ein-/Ausgabefehler
grInvalidFont	-13	*Invalid font file* Font-Datei ungültig
grInvalidFontNum	-14	*Invalid font number* Kennziffer für Font nicht definiert
grInvalidDeviceNum	-15	*Invalid device number* Kennziffer für Grafiktreiber nicht definiert
grInvalidVersion	-18	*Invalid version number* Version wird nicht unterstützt

Der Aufruf von *graphresult* setzt den Statuscode automatisch auf 0 zurück. Prüfungen des Statuscodes sollten deshalb über eine temporäre Variable erfolgen.

Rückgabewert *graphresult* liefert die aktuelle Fehlernummer des Grafikpakets (mögliche Werte von -15 bis 0) zurück, *grapherrormsg* einen Zeiger auf einen String mit dem Text der von *graphresult* gelieferten Meldung.

Siehe auch *detectgraph*, *drawpoly*, *fillpoly*, *floodfill*, *grapherrormsg*, *initgraph*, *pieslice*, *registerbgidriver*, *registerbgifont*, *setallpalette*, *setcolor*, *setfillstyle*, *setgraphmode*, *setlinestyle*, *setpalette*, *settextjustify*, *settextstyle*, *setusercharsize*, *setviewport*, *setvisualpage*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
```

```
   errorcode = graphresult();

   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

harderr, hardresume, hardretn

Funktion

Installiert eine Routine zur Bearbeitung von Hardware-Fehlern und ruft diese auf.

Syntax

```
#include <dos.h>
void harderr(int (*handler)());
void hardresume(int axret);
void hardretn(int retn);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

Die von *harderr* gesetzte Fehlerbehandlungsroutine kann *hardresume* aufrufen, um zu DOS zurückzukehren. Der von *hardresume* zurückgelieferte Ergebniscode *rescode* enthält entweder Abort (2), Retry (1) oder Ignore (0). Der Abbruch wird über einen Aufruf des DOS-Interrupts 0x23 (Strg-Untbr) durchgeführt.

Die von *harderr* installierte Fehlerbehandlungsroutine kann mit einem Aufruf von *hardretn* direkt zum Anwendungsprogramm zurückkehren. Der Rückgabewert der DOS-Funktion, die den Hardware-Fehler auslöst, ist der Wert, der an *hardretn* übergeben wurde.

harderr installiert für das aktuelle Programm eine Hardware-Fehlerbehandlungsroutine. Diese Routine wird bei jedem Auftreten eines Interrupt 0x24 aufgerufen. Eine Beschreibung dieses Interrupts finden Sie in Ihrem DOS-Referenzhandbuch.

Die Funktion, auf die *handler* zeigt, wird beim Auftreten eines solchen Interrupts mit den folgenden Argumenten aufgerufen:

```
handler(int errval, int ax, int bp, int si);
```

errval ist der von DOS im DI-Register übergebene Fehlercode. *ax*, *bp* und *si* enthalten die Werte der Register AX, BP und SI:

- über den Wert von *ax* wird zwischen Gerätefehlern (*ax = 0*) und Laufwerksfehlern (*ax <> 0*) unterschieden. Wenn *ax* einen Wert ungleich Null hat, dann ergibt die Operation *ax & 0x00FF* die Nummer des entsprechenden Laufwerks (A = 0, B = 1, etc.);
- die Parameter bp und si zeigen zusammen auf den Header des Gerätetreibers, in dem der Fehler aufgetreten ist bzw. bemerkt wurde. *bp* enthält die Segmentadresse, *si* den Offset.

Die Funktion, auf die *handler* zeigt, wird nicht direkt aufgerufen. *harderr* installiert einen DOS-Interrupt-Handler, der diese Funktion aufruft.

Die DOS-Funktionen 0x01 bis 0x0C stehen auch innerhalb der Fehlerbehandlungsroutine zur Verfügung – alle anderen Aufrufe von DOS bringen dagegen das System zum Absturz. Insbesondere die Standardfunktionen von C zur Ein- und Ausgabe sowie die UNIX-Emulationen dürfen nicht verwendet werden.

Der Rückgabewert von *handler* muß 0 für »Ignorieren«, 1 für »Wiederholen« und 2 für »Abbrechen« sein.

Rückgabewert Keiner.

Siehe auch *peek, poke*

Beispiel

```
/*
This program will trap disk errors and
prompt the user for action. Try running it
with no disk in drive A: to invoke its
functions.
*/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define IGNORE  0
#define RETRY   1
#define ABORT   2

int buf[500];
```

```
/*
define the error messages for trapping disk problems
*/
static char *err_msg[] = {
    "write protect",
    "unknown unit",
    "drive not ready",
    "unknown command",
    "data error (CRC)",
    "bad request",
    "seek error",
    "unknown media type",
    "sector not found",
    "printer out of paper",
    "write fault",
    "read fault",
    "general failure",
    "reserved",
    "reserved",
    "invalid disk change"
};

error_win(char *msg)
{
   int retval;

   cputs(msg);

/*
prompt for user to press a key to abort, retry, ignore
*/
   while(1)
   {
     retval= getch();
     if (retval == 'a' || retval == 'A')
     {
       retval = ABORT;
       break;
     }
     if (retval == 'r' || retval == 'R')
     {
       retval = RETRY;
       break;
     }
     if (retval == 'i' || retval == 'I')
     {
       retval = IGNORE;
       break;
```

```
      }
   }

   return(retval);
}

/*
pragma warn -par reduces warnings which occur
due to the non use of the parameters errval,
bp and si to the handler.
*/
#pragma warn -par

int handler(int errval,int ax,int bp,int si)
{
   static char msg[80];
   unsigned di;
   int drive;
   int errorno;

   di= _DI;
/*
if this is not a disk error then it was
another device having trouble
*/

   if (ax < 0)
   {
      /* report the error */
      error_win("Device error");
      /* and return to the program directly
         requesting abort
      */
      hardretn(ABORT);
   }
/* otherwise it was a disk error */
   drive = ax & 0x00FF;
   errorno = di & 0x00FF;
/* report which error it was */
   sprintf(msg, "Error: %s on drive %c\r\n"
           "A)bort, R)etry, I)gnore: ",
           err_msg[errorno], 'A' + drive);
/*
   return to the program via dos interrupt 0x23
   with abort, retry, or ignore as input by the user.
*/
   hardresume(error_win(msg));
   return ABORT;
```

```
}
#pragma warn +par

int main(void)
{
/*
install our handler on the hardware problem interrupt
*/
   harderr(handler);
   clrscr();
   printf("Make sure there is no disk in drive A:\n");
   printf("Press any key ....\n");
   getch();
   printf("Trying to access drive A:\n");
   printf("fopen returned %p\n",
          fopen("A:temp.dat", "w"));
   return 0;
}
```

_harderr

Funktion

Installiert eine Routine zur Behandlung von Hardware-Fehlern.

Syntax

```
#include <dos.h>
void _harderr(int (far *handler)());
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

_harderr installiert für das aktuelle Programm eine Hardware-Fehlerbehandlungsroutine. Diese Routine wird bei jedem Auftreten eines Interrupt 0x24 aufgerufen. Eine Beschreibung dieses Interrupt finden Sie in Ihrem DOS-Referenzhandbuch.

Die Funktion, auf die *handler* zeigt, wird beim Auftreten eines solchen Interrupt mit den folgenden Argumenten aufgerufen:

```
void far handler(unsigned deverr, unsigned errval,
unsigned far *devhdr);
```

- *deverr* ist der von DOS im AX-Register übergebene Geräte-Fehlercode;
- *errval* ist der von DOS im DI-Register übergebene Fehlercode;
- *devhdr* ist ein **far**-Zeiger auf den Header des Gerätetreibers, in dem der Fehler aufgetreten ist bzw. bemerkt wurde (er wird von DOS im Registerpaar BP:SI übergeben).

Beziehen Sie sich in der Fehlerbehandlungsroutine nicht direkt auf CPU-Register, sondern verwenden Sie die oben aufgeführten Argumente.

Über den Wert von *deverr* wird zwischen Geräte- und Diskettenfehlern unterschieden. Wenn das Bit 15 von *deverr* 0 ist, hat sich ein Diskettenfehler ereignet, andernfalls handelt es sich um einen Gerätefehler. Bei einem Diskettenfehler ergibt die Operation *deverr & 0x00FF* die Nummer des entsprechenden Laufwerks (A = 0, B = 1, etc.).

Die Funktion, auf die *handler* zeigt, wird nicht direkt aufgerufen. *_harderr* installiert einen DOS-Interrupt-Handler, der diese Funktion aufruft.

Die DOS-Funktionen 0x01 bis 0x0C stehen auch innerhalb der Fehlerbehandlungsroutine zur Verfügung – alle anderen Aufrufe von DOS bringen dagegen das System zum Absturz. Insbesondere die Standardfunktionen von C zur Ein- und Ausgabe sowie die UNIX-Emulationen dürfen nicht verwendet werden. Die Fehlerbehandlungsroutine kann entweder über *_hardresume* oder über *_hardretn* beendet werden.

Rückgabewert Keiner.

Siehe auch *_hardresume*, *_hardretn*

Beispiel

```
/* This program traps disk errors and
   prompts the user for action.
   Try running it with no disk in drive A
   to invoke its functions.
*/

#include <stdio.h>
#include <ctype.h>
#include <dos.h>
#include <fcntl.h>

int buf[500];

/* define the error messages for
   trapping disk problems
*/
static char *err_msg[] =
{
  "write protect",   "unknown unit",
  "drive not ready", "unknown command",
  "data error (CRC)","bad request",
  "seek error",      "unknown media type",
  "sector not found","printer out of paper",
  "write fault",     "read fault",
  "general failure", "reserved",
  "reserved",        "invalid disk change"
};
```

```
static void mesg(char *s)
{
  while (*s)
    bdos(2,*s++,0);
}

static int getkey(void)
{
  return (bdos(7, 0, 0) & 0xff);
}

error_win(char *msg)
{
  int c;

  /* prompt user to press a key to
     abort, retry, ignore, fail
  */
  while(1)
  {
    mesg(msg);
    c = tolower(getkey());
    mesg("\r\n");
    switch (c)
    {
    case 'a':
      return (_HARDERR_ABORT);
    case 'r':
      return (_HARDERR_RETRY);
    case 'i':
      return (_HARDERR_IGNORE);
    case 'f':
      return (_HARDERR_FAIL);
    }
  }
}

/* pragma warn -par reduces warnings which occur
   due to the nonuse of the parameter devhdr
*/
#pragma warn -par

void far handler(unsigned deverr, unsigned errval,
                 unsigned far *devhdr)
{
  static char msg[80];
  int drive;
  int errorno;

  /* if this not disk error then another device
     having trouble
```

```
  */
  if (deverr & 0x8000) {
    error_win("Device error");  /* report the error */

 /* return to the program directly requesting abort */
    _hardretn(5);  /* 5 = DOS "access denied" error */
  }
  drive = deverr & 0x00FF;
                  /* otherwise it was disk error */
  errorno = errval & 0x00FF;

  /* report which error it was */
  sprintf(msg, "Error: %s on drive %c\r\n"
          "A)bort, R)etry, I)gnore, F)ail: ",
    err_msg[errorno], 'A' + drive);

  /* return to program via dos interrupt 0x23
     with abort, retry or ignore as input by the user
  */
  _hardresume(error_win(msg));
}

#pragma warn +par

int main(void)
{
  int handle;

  /* install our handler on the
     hardware problem interrupt
  */
  _harderr(handler);
  printf("Make sure there is no disk in drive A:\n");
  printf("Press any key ....\n");
  getkey();
  printf("Trying to access drive A:\n");
  printf("_dos_open returned 0x%x\n",
    _dos_open("A:temp.dat", O_RDONLY, &handle));
  return 0;
}
```

_hardresume

Funktion Hardware-Fehlerbehandlungsroutine.

Syntax
```
#include <dos.h>
void _hardresume(int rescode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Die von *_harderr* gesetzte Fehlerbehandlungsroutine kann *_hardresume* aufrufen, um zu DOS zurückzukehren.

Der von *_hardresume* zurückgelieferte Ergebniscode *rescode* enthält einen der folgenden Werte:

_HARDERR_ABORT	Abbruch wird über einen Aufruf des DOS-Interrupt 0x23 (Strg-Untbr) durchgeführt.
_HARDERR_IGNORE	Fehler wird ignoriert.
_HARDERR_RETRY	Operation wird wiederholt.
_HARDERR_FAIL	Operation erfolglos.

Rückgabewert *_hardresume* hat keinen Rückgabewert und kehrt nicht zur aufrufenden Funktion zurück.

Siehe auch *_harderr*, *_hardretn*

Beispiel Siehe das Beispiel für *_harderr*.

_hardretn

Funktion Hardware-Fehlerbehandlungsroutine.

Syntax
```
#include <dos.h>
void _hardretn(int retn);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Die von *_harderr* installierte Fehlerbehandlungsroutine kann mit einem Aufruf von *_hardretn* direkt zum Anwendungsprogramm zurückkehren.

DOS-Funktionen unter 0x38, die einen Hardware-Fehler auslösen und die Fehlerbedingung anzeigen, setzen bei der Rückkehr zum Anwendungsprogramm das Register AL auf 0xFF.

Das Argument *retn* wird bei allen DOS-Funktionen unter 0x38 ignoriert.

Für DOS-Funktionen größer oder gleich 0x38 sollte das Argument *retn* ein DOS-Fehlercode sein. Er wird bei der Rückkehr zum Anwendungsprogramm im AX-Register übergeben. Das Carry-Flag ist gesetzt und zeigt an, daß die Operation einen Fehler zur Folge hatte.

Rückgabewert *_hardretn* hat keinen Rückgabewert und kehrt nicht zur aufrufenden Funktion zurück.

Siehe auch *_harderr*, *_hardresume*

Beispiel Siehe das Beispiel für *_harderr*.

heapcheck

Funktion Überprüft den Heap.

Syntax
```
#include <alloc.h>
int heapcheck(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *heapcheck* überprüft jeden Block des Heap auf seine Zeiger, Größe und weitere wichtige Attribute. In den großen Speichermodellen wird *heapcheck* durch einen Aufruf von *farheapcheck* ersetzt.

Rückgabewert Bei fehlerfreier Ausführung wird ein Wert größer als Null, im Fall eines Fehlers ein Wert kleiner als Null zurückgeliefert. Im einzelnen können von der Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen Heap (Wert 1)
_HEAPOK	Kein Fehler gefunden (Wert 2)
_HEAPCORRUPT	Der Heap ist ungültig (Wert -1)

Siehe auch *farheapcheck*

Beispiel
```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   if( heapcheck() == _HEAPCORRUPT )
      printf( "Heap is corrupted.\n" );
   else
      printf( "Heap is OK.\n" );

   return 0;
}
```

heapcheckfree

Funktion

Überprüft die freien Blöcke des Heap auf einen konstanten Wert.

Syntax

```
#include <alloc.h>
int heapcheckfree(unsigned int fillvalue);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

heapcheckfree überprüft die freien Blöcke des Heap auf einen konstanten Wert.

Rückgabewert

Bei erfolgreicher Ausführung ist der Rückgabewert größer als Null, im Fall eines Fehlers kleiner als Null. Im einzelnen können von der Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen Heap (Wert 1)
_HEAPOK	Kein Fehler gefunden (Wert 2)
_HEAPCORRUPT	Der Heap ist ungültig (Wert -1)
_BADVALUE	Ein anderer Wert als der Füllwert wurde gefunden (Wert -3)

Siehe auch

farheapcheckfree

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <mem.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;
   int res;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   if( heapfillfree( 1 ) < 0 )
   {
      printf( "Heap corrupted.\n" );
      return 1;
```

```
    }

    for( i = 1; i < NUM_PTRS; i += 2 )
       memset( array[ i ], 0, NUM_BYTES );

    res = heapcheckfree( 1 );
    if( res < 0 )
       switch( res )
       {
          case _HEAPCORRUPT:
               printf( "Heap corrupted.\n" );
               return 1;
          case _BADVALUE:
               printf( "Bad value in free space.\n" );
               return 1;
          default:
               printf( "Unknown error.\n" );
               return 1;
       }

    printf( "Test successful.\n" );
    return 0;
}
```

heapchecknode

Funktion

Überprüft einen einzelnen Knoten des Heap.

Syntax

```
#include <alloc.h>
int heapchecknode(void *node);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

heapchecknode überprüft die einzelnen Knoten auf dem Heap. Wird *heapchecknode* nach der Freigabe eines Knotens mit einem Zeiger auf diesen Block aufgerufen, kann _BADNODE anstelle des erwarteten _FREEENTRY zurückgeliefert werden. Der Grund liegt darin, daß benachbarte freie Blöcke auf dem Heap zusammengefaßt werden und der betreffende Block deshalb nicht mehr existiert.

Rückgabewert

Bei erfolgreicher Ausführung ist der Rückgabewert größer als Null, im Fall eines Fehlers kleiner als Null. Im einzelnen können von der Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen Heap (Wert 1)
_HEAPCORRUPT	Der Heap ist ungültig (Wert -1)

_BADNODE	Der Knoten wurde nicht gefunden (Wert -2)
_FREEENTRY	Der Knoten ist ein freier Block (Wert 3)
_USEDENTRY	Der Knoten ist ein belegter Block (Wert 4)

Siehe auch *farheapchecknode*

Beispiel

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   for( i = 0; i < NUM_PTRS; i++ )
   {
      printf( "Node %2d ", i );
      switch( heapchecknode( array[ i ] ) )
      {
         case _HEAPEMPTY:
              printf( "No heap.\n" );
              break;
         case _HEAPCORRUPT:
              printf( "Heap corrupt.\n" );
              break;
         case _BADNODE:
              printf( "Bad node.\n" );
              break;
         case _FREEENTRY:
              printf( "Free entry.\n" );
              break;
         case _USEDENTRY:
              printf( "Used entry.\n" );
              break;
         default:
              printf( "Unknown return code.\n" );
               break;
      }
   }

   return 0;
}
```

heapfillfree

Funktion — Füllt die freien Blöcke des Heap mit einem konstanten Wert.

Syntax

```
#include <alloc.h>
int heapfillfree(unsigned int fillvalue);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *heapfillfree* füllt die freien Blöcke des Heap mit einem konstanten Wert.

Rückgabewert — Bei erfolgreicher Ausführung ist der Rückgabewert größer als Null, im Fall eines Fehlers kleiner als Null. Im einzelnen können von der Funktion folgende Werte zurückgeliefert werden:

_HEAPEMPTY	Es gibt keinen Heap (Wert 1)
_HEAPOK	Kein Fehler gefunden (Wert 2)
_HEAPCORRUPT	Der Heap ist ungültig (Wert -1)

Siehe auch — *farheapfillfree*

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <mem.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;
   int res;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   if( heapfillfree( 1 ) < 0 )
   {
      printf( "Heap corrupted.\n" );
      return 1;
   }

   for( i = 1; i < NUM_PTRS; i += 2 )
```

```
        memset( array[ i ], 0, NUM_BYTES );

    res = heapcheckfree( 1 );
    if( res < 0 )
        switch( res )
        {
            case _HEAPCORRUPT:
                printf( "Heap corrupted.\n" );
                return 1;
            case _BADVALUE:
                printf( "Bad value in free space.\n" );
                return 1;
            default:
                printf( "Unknown error.\n" );
                return 1;
        }

    printf( "Test successful.\n" );
    return 0;
}
```

heapwalk

Funktion Durchwandert den Heap Knoten für Knoten.

Syntax
```
#include <alloc.h>
int heapwalk(struct heapinfo *hi);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *heapwalk* durchwandert den Heap Knoten für Knoten. Die Funktion geht davon aus, daß der Heap gültig ist (überprüfen Sie dies vorher mit *heapcheck*). Mit dem letzten Block wird _HEAPOK zurückgeliefert. Bei einem erneuten Aufruf von *heapwalk* ist das Ergebnis _HEAPEND.

heapwalk erhält einen Zeiger auf eine Struktur vom Typ *heapinfo* (deklariert in alloc.h). Setzen Sie das Feld *hi.ptr* beim ersten Aufruf der Funktion auf NULL. Nach der Rückkehr enthält *hi.ptr* die Adresse des ersten Blocks, während *hi.size* die Größe des Blocks in Bytes angibt. Wird der Block aktuell benutzt, ist das Flag *hi.in_use* gesetzt.

Rückgabewert *heapwalk* kann folgende Werte zurückliefern:

_HEAPEMPTY	Es gibt keinen Heap (Wert 1)
_HEAPOK	Der *heapinfo*-Block enthält gültige Daten (Wert 2)
_HEAPEND	Das Ende des Heap wurde erreicht (Wert 5)

Siehe auch *farheapwalk*

Beispiel

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main( void )
{
   struct heapinfo hi;
   char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   hi.ptr = NULL;
   printf( "   Size   Status\n" );
   printf( "   ----   ------\n" );
   while( heapwalk( &hi ) == _HEAPOK )
   printf( "%7u    %s\n", hi.size,
          hi.in_use ? "used" : "free" );

   return 0;
}
```

highvideo

Funktion Setzt hohe Intensität für nachfolgende Ausgaben von Textzeichen.

Syntax

```
#include <conio.h>
void highvideo(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *highvideo* setzt das Bit für »hohe Intensität« der aktuellen Vordergrundfarbe und damit »hohe Intensität« für folgende Textausgaben.

Bereits ausgegebene Zeichen bleiben unverändert – es wird lediglich das Zeichenattribut für Funktionen wie z.B. *cprintf* festgelegt, die direkte Bildschirmausgaben durchführen, nachdem *highvido* aufgerufen wurde.

Rückgabewert Keiner.

Siehe auch *cprintf, cputs, gettextinfo, lowvideo, normvideo, textattr, textcolor*

Beispiel

```
#include <conio.h>

int main(void)
{
   clrscr();

   lowvideo();
   cprintf("Low Intensity text\r\n");
   highvideo();
   gotoxy(1,2);
   cprintf("High Intensity Text\r\n");

   return 0;
}
```

hypot, hypotl

Funktion Berechnet die Hypotenuse eines rechtwinkligen Dreiecks.

Syntax

```
#include <math.h>
double hypot(double x, double y);
long double hypotl(long double x, long double y);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x			*hypot*
x		x			*hypotl*

Beschreibung *hypot* berechnet den Wert z, hierbei gilt: $z^2 = x^2 + y^2$ sowie $z \geq 0$.

Dies entspricht der Länge der Hypotenuse eines rechtwinkligen Dreiecks, dessen Kathete und Ankathete die Längen x bzw. y haben.

hypotl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert Bei fehlerfreier Ausführung wird z als **double** (*hypot*) bzw. **long double** (*hypotl*) zurückgeliefert. Im Falle eines Fehlers (zum Beispiel bei einem Überlauf) liefert *hypot* HUGE_VAL und *hypotl* _LHUGE_VAL zurück, und die globale Variable *errno* wird auf folgenden Wert gesetzt:

ERANGE Ergebnis außerhalb des gültigen Bereichs

Die Fehlerbehandlung für *hypot* und *hypotl* kann über die Funktion *matherr* bzw. *_matherrl* verändert werden.

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 3.0;
   double y = 4.0;

   result = hypot(x, y);
   printf("The hypotenuse is: %lf\n", result);

   return 0;
}
```

imag

Funktion Liefert den imaginären Teil einer komplexen Zahl.

Syntax

```
#include <complex.h>
double imag(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung Die zu komplexen Zahlen gehörigen Daten bestehen aus zwei (**double**) Gleitkommazahlen. *imag* liefert diejenige Zahl, die als imaginärer Teil betrachtet wird. *real* liefert diejenige Zahl, die als reeller Teil betrachtet wird.

Rückgabewert *imag* liefert den imaginären Teil einer komplexen Zahl.

Siehe auch *complex, conj, real*

Beispiel

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary real part = "
        << imag(z) << "\n";
   cout << "z has complex conjugate = "
        << conj(z) << "\n";
   return 0;
}
```

imagesize

Funktion
Berechnet die Anzahl der Bytes, die zur Speicherung grafischer Bildausschnitte benötigt werden.

Syntax
```
#include <graphics.h>
unsigned far imagesize(int left, int top, int right,
                       int bottom);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung
imagesize berechnet die Anzahl von Bytes, die zur Speicherung eines Bildausschnitts benötigt werden. Ist die zur Speicherung benötigte Anzahl von Bytes größer als oder gleich *64 KByte - 1 Byte*, liefert *imagesize* den Wert 0xFFFF (-1) zurück.

Rückgabewert
imagesize liefert die Größe des angegebenen Bildausschnitts in Bytes zurück.

Siehe auch
getimage, putimage

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
```

```
   maxx = getmaxx();
   x = 0;
   y = getmaxy() / 2;

   /* draw the image to be grabbed */
   draw_arrow(x, y);

   /* calculate the size of the image */
   size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
                    y+ARROW_SIZE);

   /* allocate memory to hold the image */
   arrow = malloc(size);

   /* grab the image */
   getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
            y+ARROW_SIZE, arrow);

   /* repeat until a key is pressed */
   while (!kbhit())
   {
      /* erase old image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);

      x += ARROW_SIZE;
      if (x >= maxx)
          x = 0;

      /* plot new image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
   }

   /* clean up */
   free(arrow);
   closegraph();
   return 0;
}

void draw_arrow(int x, int y)
{
   /* draw an arrow on the screen */
   moveto(x, y);
   linerel(4*ARROW_SIZE, 0);
   linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
   linerel(0, 2*ARROW_SIZE);
   linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
```

initgraph

Funktion Initialisiert das Grafiksystem.

Syntax
```
#include <graphics.h>
void far initgraph(int far *graphdriver,
                   int far *graphmode,
                   char far *pathtodriver);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Die Funktion *initgraph* initialisiert das Grafiksystem, indem sie einen Grafiktreiber von der Platte lädt (bzw. einen mit *registerbgidriver* eingebundenen Treiber überprüft) und in den Grafikmodus schaltet. Um das Grafiksystem zu starten, rufen Sie zuerst *initgraph* auf. *initgraph* lädt den Grafiktreiber und schaltet das System in den Grafikmodus, den Sie angeben.

Sie können *initgraph* auch anweisen, die angeschlossene Grafikkarte zu untersuchen und einen entsprechenden Treiber auszuwählen.

Falls Sie *initgraph* zur automatischen Erkennung veranlassen, werden Grafiktreiber und Grafikmodus durch einen Aufruf von *detectgraph* ausgewählt. *initgraph* wählt für alle Einstellungen des Grafiksystems die Voreinstellungen (aktuelle Position (CP), Farbpalette, Farbe, Zeichenfenster, usw.) und setzt *graphresult* auf 0. Normalerweise lädt *initgraph* einen Grafiktreiber, indem über einen Aufruf von *_graphgetmem* Speicher für den Treiber belegt und die entsprechende .BGI-Datei von der Platte geladen wird. Alternativ zu diesem dynamischen Verfahren können Sie auch eine Grafiktreiberdatei (oder mehrere davon) direkt in die ausführbare Datei einbinden.

pathtodriver ist ein Zeiger auf einen String, der den Suchweg für Grafiktreiberdateien festlegt: *initgraph* sucht zuerst im angegebenen Verzeichnis, danach nötigenfalls im aktuellen Verzeichnis. Wenn *pathtodriver* NULL ist, findet die Suche nur im aktuellen Verzeichnis statt. Dieser Suchweg bestimmt auch das Verzeichnis, in dem *settextstyle* nach Vektor-Font-Dateien (*.CHR) sucht.

Für **graphdriver* – die Kennziffer des Grafiktreibers – sind die folgenden Konstanten des Aufzählungstyps *graphics_drivers* in graphics.h definiert:

Name	Wert	
DETECT	0	(automatische Erkennung)
CGA	1	
MCGA	2	

Name	Wert
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

**graphmode* ist ein Integer, der den Grafikmodus für die Initialisierung festlegt (außer, wenn für **graphdriver* DETECT übergeben wird; in diesem Fall wählt *initgraph* die höchstmögliche Auflösung für den gefundenen Treiber).

Für *graphmode* – den zu setzenden Grafikmodus – sind die folgenden Konstanten in graphics.h definiert:

Grafikmodus	Wert	Spalten * Zeilen	Palette	Seiten
Grafiktreiber CGA				
CGAC0	0	320*200	C0	1
CGAC1	1	320*200	C1	1
CGAC2	2	320*200	C2	1
CGAC3	3	320*200	C3	1
CGAHI	4	640*200	2 Farben	1
Grafiktreiber MCGA				
MCGAC0	0	320*200	C0	1
MCGAC1	1	320*200	C1	1
MCGAC2	2	320*200	C2	1
MCGAC3	3	320*200	C3	1
MCGAMED	4	640*200	2 Farben	1
MCGAHI	5	640*480	2 Farben	1
Grafiktreiber EGA				
EGALO	0	640*200	16 Farben	4
EGAHI	1	640*350	16 Farben	2
Grafiktreiber EGA64				
EGA64LO	0	640*200	16 Farben	1
EGA64HI	1	640*350	4 Farben	1
Grafiktreiber EGAMONO				
EGAMONOHI	3	640*350	2 Farben	1*
EGAMONOHI	3	640*350	2 Farben	2**
Grafiktreiber HERC				
HERCMONOHI	0	720*348	2 Farben	2

Grafikmodus	Wert	Spalten * Zeilen	Palette	Seiten
Grafiktreiber ATT400				
ATT400C0	0	320*200	C0	1
ATT400C1	1	320*200	C1	1
ATT400C2	2	320*200	C2	1
ATT400C3	3	320*200	C3	1
ATT400MED	4	640*200	2 Farben	1
ATT400HI	5	640*400	2 Farben	1
Grafiktreiber VGA				
VGALO	0	640*200	16 Farben	2
VGAMED	1	640*350	16 Farben	2
VGAHI	2	640*480	16 Farben	1
Grafiktreiber PC3270				
PC3270HI	0	720*350	2 Farben	1
Grafiktreiber IBM8514				
IBM8514LO	0	640*480	256 Farben	
IBM8514HI	1	1024*768	256 Farben	

* 64 KByte auf monochroner EGA-Karte
** 256 KByte auf monochromer EGA-Karte

Hinweis: Sie müssen für **graphdriver* und * *graphmode* gültige Werte aus den beiden obenstehenden Tabellen einsetzen, wenn sich Ihr Programm nicht unvorhersehbar verhalten soll. Nur für **graphdriver* = DETECT ist der Wert von **graphmode* unerheblich.

In den Farbpaletten ...C0 bis ...C3 (d.h. den CGA-Karten und Kompatiblen) kann nur die Farbe des Hintergrundes (Eintrag 0) verändert werden. Die restlichen Farben sind durch die jeweilige Palette festgelegt. Diese Paletten sind detailliert in Kapitel 11 »Text und Grafik« des *Programmierhandbuchs* beschrieben und in der folgenden Tabelle zusammengefaßt:

Palette	Pixelwert		
	1	2	3
C0	hellgrün	hellrot	gelb
C1	helltürkis	hell-violett	weiß
C2	grün	rot	braun
C3	türkis	violett	hellgrau

Nach einem Aufruf von *initgraph* wird **graphdriver* auf den aktuellen Grafiktreiber und **graphmode* auf den aktuellen Grafikmodus gesetzt.

Rückgabewert *initgraph* setzt immer den internen Fehlercode. Bei erfolgreicher Ausführung wird der Code auf 0 gesetzt. Wenn *initgraph*

nicht fehlerfrei ausgeführt werden kann, bekommt **graphdriver* einen der folgenden Werte zugewiesen, die auch von *graphresult* zurückgeliefert werden:

Name	Wert	Text/Beschreibung
grNotDetected	-2	Kein grafikfähiger Adapter gefunden
grFileNotFound	-3	Grafiktreiberdatei nicht gefunden
grInvalidDriver	-4	Grafiktreiberdatei ungültig
grNoLoadMem	-5	Nicht genug Platz im Hauptspeicher für den Grafiktreiber

Siehe auch *closegraph, detectgraph, getdefaultpalette, getdrivername, getgraphmode, getmoderange, graphdefaults, _graphgetmem, graphresult, installuserdriver, registerbgidriver, registerbgifont, restorecrtmode, setgraphbufsize, setgraphmode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);             /* return with error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

inp

Funktion Liest ein Byte aus einer Schnittstelle.

Syntax
```
#include <conio.h>
int inp(unsigned portid);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Das Makro *inp* liest ein einzelnes Byte von der durch *portid* angegebenen Eingabeschnittstelle.

inp ist zusätzlich als inline-Makro definiert – der Compiler verwendet nur dann »echte« Funktionsaufrufe, wenn conio.h nicht aufgenommen wird (oder ein entsprechendes #*undef* stattgefunden hat).

Rückgabewert *inp* liefert den gelesenen Wert zurück.

Siehe auch *inpw*, *outp*, *outpw*

Beispiel
```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   int result;
   int port = 0;  /* serial port 0 */

   result = inp(port);
   printf("Word read from port %d = 0x%X\n",
          port, result);
   return 0;
}
```

inport

Funktion Liest ein WORD aus einer Schnittstelle.

Syntax
```
#include <dos.h>
int inport(int portid);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *inport* liest das niederwertige Byte eines WORD aus der durch *portid* angegebenen Schnittstelle und das höherwertige Byte von der durch *portid +1* angegebenen Adresse. Diese Funktion arbeitet wie die 80x86-Anweisung *IN*.

Rückgabewert *inport* liefert den gelesenen Wert zurück.

Siehe auch *inportb, outport, outportb*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  int result;
  int port = 0;
  result = inport(port);
  printf("Word read from port %d = 0x%X\n",
         port, result);
  return 0;
}
```

inportb

Funktion Liest ein Byte aus einer Schnittstelle.

Syntax

```
#include <dos.h>
unsigned char inportb(int portid);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung *inportb* liest ein einzelnes Byte von der durch *portid* angegebenen Schnittstelle.

inportb ist zusätzlich als inline-Makro definiert – der Compiler verwendet nur dann »echte« Funktionsaufrufe, wenn dos.h nicht aufgenommen wird oder wenn ein entsprechendes #*undef* stattgefunden hat.

Rückgabewert *inportb* liefert den gelesenen Wert zurück.

Siehe auch *inport, outport, outportb*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   unsigned char result;
   int port = 0;           /* serial port 1 */

   result = inportb(port);
   printf("Byte read from port %d = 0x%X\n",
          port, result);
   return 0;
}
```

inpw

Funktion — Liest ein WORD aus einer Schnittstelle.

Syntax

```
#include <conio.h>
unsigned inpw(unsigned portid);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — Das Makro *inpw* liest ein 16-Bit-WORD von der durch *portid* angegebenen I/O-Adresse. *inpw* liest das niederwertige Byte des WORD von *portid* und das höherwertige von *portid +1*.

inpw ist zusätzlich als inline-Makro definiert – der Compiler verwendet nur dann »echte« Funktionsaufrufe, wenn conio.h nicht aufgenommen wird oder ein entsprechendes #*undef* stattgefunden hat.

Rückgabewert — *inpw* liefert den gelesenen Wert zurück.

Siehe auch — *inp, outp, outpw*

Beispiel

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
  unsigned result;
  unsigned port = 0;
  result = inpw(port);
  printf("Word read from port %d = 0x%X\n",
         port, result);
  return 0;
}
```

insline

Funktion — Fügt eine Leerzeile an der Position des Cursors in das Textfenster ein.

Syntax

```
#include <conio.h>
void insline(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *insline* verschiebt alle Zeilen zwischen der aktuellen Position des Cursors und dem Ende des Textfensters um eine Zeile nach unten; die unterste Zeile wird dabei aus dem Fenster geschoben.

Die Zeile, auf der sich der Cursor befindet, wird mit der für den Hintergrund gesetzten Farbe gefüllt.

insline arbeitet nur im Textmodus.

Rückgabewert Keiner.

Siehe auch *clreol*, *delline*, *window*

Beispiel

```
#include <conio.h>

int main(void)
{
   clrscr();
   cprintf("INSLINE inserts an empty line in "
           "the text window\r\n");
   cprintf("at the cursor position using the "
           "current text\r\n");
   cprintf("background color.  All lines below "
           "the empty one\r\n");
   cprintf("move down one line and the bottom "
           "line scrolls\r\n");
   cprintf("off the bottom of the window.\r\n");
   cprintf("\r\nPress any key to continue:");
   gotoxy(1, 3);
   getch();
   insline();
   getch();
   return 0;
}
```

installuserdriver

Funktion Fügt einen neuen Gerätetreiber in die BGI-Gerätetreibertabelle ein.

Syntax

```
#include graphics.h
int far installuserdriver(char far *name,
                          int huge (*detect)(void));
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Mit *installuserdriver* können Sie einen neu erworbenen Gerätetreiber in die interne BGI-Tabelle einfügen. Der Parameter *name* gibt den Namen der neuen Gerätetreiberdatei an (.BGI) und *detect* steht für einen Zeiger auf eine optionale automatische Erkennungsfunktion, die zu dem neuen Treiber mitgeliefert sein kann. Diese Erkennungsfunktion hat keine Parameter und liefert einen Integerwert zurück.

Es gibt zwei Möglichkeiten für den Einsatz des neuen Treibers. Nehmen wir an, Sie haben eine neue Grafikkarte mit dem Namen *Spiffy Graphics Array* (*SGA*) erworben und der Hersteller liefert dazu einen BGI-Gerätetreiber (SGA.BGI). Sie können nun ganz einfach diesen Treiber durch Aufruf von *installuserdriver* installieren und den Rückgabewert (die zugewiesene Treibernummer) direkt an *initgraph* übergeben.

Die zweite (allgemeinere) Möglichkeit, den Treiber zu benutzen, besteht darin, eine automatische, von *initgraph* als Bestandteil der Hardware-Erkennungs-Routine aufzurufende, Erkennungsfunktion einzubinden. Voraussetzung ist, daß der Hersteller der SGA diese automatische Erkennungsfunktion mitliefert. Wenn Sie durch Aufruf von *installuserdriver* den Treiber installieren, übergeben Sie die Adresse dieser Funktion zusammen mit dem Namen der Gerätetreiberdatei.

Nach der Übergabe des Geätetreiberdateinamens und der automatischen SGA-Erkennungsfunktion rufen Sie *initgraph* auf und lassen den automatischen Erkennungsprozeß ablaufen. Bevor *initgraph* seine eigene automatische Erkennungsfunktion (*detectgraph*) aufruft, wird zuerst die SGA-Erkennungsfunktion aufgerufen. Kann die automatische SGA-Erkennungsfunktion die SGA-Hardware nicht finden, liefert sie den Wert -11 (grError) zurück und *initgraph* führt dann den normalen Hardware-Erkennungsprozeß durch, der den Aufruf weiterer vom Hersteller gelieferten Erkennungsfunktionen beinhalten kann. Weitere vorhandene Funktionen werden in der Reihenfolge aufgerufen, in der sie »installiert« wurden. Stellt die automatische SGA-Erkennungsfunktion jedoch fest, daß eine SGA-Karte vorhanden ist, liefert sie eine positive Modusnummer zurück. *initgraph* sucht und lädt SGA.BGI, setzt die Hardware in den voreingestellten, von der Funktion empfohlenen Grafikmodus und übergibt die Steuerung wieder an Ihr Programm.

Sie können bis zu 10 Treiber gleichzeitig installieren.

Rückgabewert *installuserdriver* liefert den Integer-Wert zurück, den Sie *initgraph* übergeben müßten, wenn Sie den neu installierten Grafiktreiber manuell auswählen wollen.

Siehe auch *initgraph, registerbgidriver*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* function prototypes */
int huge detectEGA(void);
void checkerrors(void);

int main(void)
```

```
{
   int gdriver, gmode;

   /* install a user written device driver */
   gdriver = installuserdriver("EGA", detectEGA);

   /* must force use of detection routine */
   gdriver = DETECT;

   /* check for any installation errors */
   checkerrors();

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* check for any initialization errors */
   checkerrors();

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* detects EGA or VGA cards */
int huge detectEGA(void)
{
   int driver, mode, sugmode = 0;

   detectgraph(&driver, &mode);
   if ((driver == EGA) || (driver == VGA))
      /* return suggested video mode number */
      return sugmode;
   else
      /* return an error code */
      return grError;
}

/* check for and report any graphics errors */
void checkerrors(void)
{
   int errorcode;

   /* read result of last graphics operation */
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
```

```
      exit(1);
    }
}
```

installuserfont

Funktion

Zur Installation von Vektor-Fonts (*.CHR), die nicht direkt von Borland angeboten werden.

Syntax

```
include <graphics.h>
int far installuserfont(char far *name);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

name zeigt auf den Pfadnamen einer Datei mit einem Vektor-Font. Mit *installuserfont* können maximal zwanzig Fonts in einem Programm installiert werden.

Rückgabewert

installuserfont liefert einen Integer zurück, der im weiteren Programmverlauf als Kennziffer für den entsprechenden Font interpretiert wird und an *settextstyle* übergeben werden kann. Wenn die Font-Tabelle des Grafiksystems keinen weiteren Eintrag aufnehmen kann, liefert die Funktion *grError* (d.h. den Wert -11).

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* function prototype */
void checkerrors(void);

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode;
   int userfont;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* check for any initialization errors */
   checkerrors();

   /* install a user defined font file */
   userfont = installuserfont("USER.CHR");
```

```
   /* check for any installation errors */
   checkerrors();

   /* select the user font */
   settextstyle(userfont, HORIZ_DIR, 4);

   /* output some text */
   outtextxy(midx, midy, "Testing!");

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* check for and report any graphics errors */
void checkerrors(void)
{
   int errorcode;

   /* read result of last graphics operation */
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
 }
```

int86

Funktion Allgemeiner 8086-Software-Interrupt.

Syntax
```
#include <dos.h>
int int86(int intno, union REGS *inregs,
          union REGS *outregs);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *int86* führt einen Software-Interrupt durch, d.h. einen Aufruf der Routine, auf die der durch *intno* angegebene Interrupt-Vektor zeigt. Vor der Ausführung des Interrupts werden die Prozessor-Register mit den durch *inregs* definierten Werten geladen, nach der Ausführung werden die Register-Inhalte in *outregs* gespeichert.

Nach der Rückkehr vom Software-Interrupt kopiert *int86* die

aktuellen Registerwerte in *outregs*, den Status des Carry-Flags in das Feld *x.cflag* von *outregs* und das 8086-Flags-Register in das Feld *x.flags* von *outregs*. Ein gesetztes Carry-Flag zeigt im allgemeinen einen Fehler an.

Hinweis: inregs und *outregs* können auf dieselbe Struktur zeigen.

Rückgabewert *int86* liefert den Wert des Registers AX nach der Ausführung des Interrupt zurück. Wenn das Carry-Flag durch den Interrupt gesetzt wurde (*outregs->x.cflag !=0*), so wird der globalen Variablen *_doserrno* der entsprechende Fehlercode zugewiesen.

Siehe auch *bdos*, *bdosptr*, *geninterrupt*, *int86x*, *intdos*, *intdosx*, *intr*

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define VIDEO 0x10

void movetoxy(int x, int y)
{
   union REGS regs;

   regs.h.ah = 2;  /* set cursor position */
   regs.h.dh = y;
   regs.h.dl = x;
   regs.h.bh = 0;  /* video page 0 */
   int86(VIDEO, &regs, &regs);
}

int main(void)
{
   clrscr();
   movetoxy(35, 10);
   printf("Hello\n");
   return 0;
}
```

int86x

Funktion Allgemeine Software-Interrupt-Schnittstelle.

Syntax

```
#include <dos.h>
int int86x(int intno, union REGS *inregs,
           union REGS *outregs, struct SREGS *segregs);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

int86x führt einen Software-Interrupt durch, d.h. einen Aufruf der Routine, auf die der durch *intno* angegebene Interrupt-Vektor zeigt. Vor der Ausführung des Interrupt werden die Prozessor-Register mit den durch *inregs* angegebenen Werten geladen.

Zusätzlich kopiert *int86x* die Werte von *segregs->ds* und *segregs->es* in die entsprechenden Register, bevor es den Interrupt ausführt. Auf diese Weise kann ein Programm, das **far** Zeiger oder ein großes Speichermodell benutzt, angeben, welches Segment für den Software-Interrupt verwendet werden soll.

Nach der Rückkehr vom Software-Interrupt kopiert *int86x* die Registerwerte in *outregs*, den Status des Carry-Flags in das Feld *x.cflag* von *outregs* und das 8086-Flags-Register in das Feld *x.flags* von *outregs*. Anschließend stellt *int86x* den alten Inhalt des DS-Registers wieder her und speichert die Werte der Segmentregister in den Feldern *segregs->es* bzw. *segregs->ds*. Ist das Carry-Flag gesetzt, zeigt das im allgemeinen einen Fehler an.

int86x ermöglicht den Aufruf eines 8086-Software-Interrupt, der mit einem anderen Wert von DS als dem voreingestellten arbeitet und/oder ein Argument in ES benötigt.

Hinweis: inregs und *outregs* können auf dieselbe Struktur zeigen.

Rückgabewert

int86x liefert den Wert des Registers AX nach der Ausführung des Interrupt zurück. Wenn das Carry-Flag durch den Interrupt gesetzt wurde (*outregs->x.cflag !=0*), so wird der globalen Variablen *_doserrno* der entsprechende Fehlercode zugewiesen.

Siehe auch

bdos, *bdosptr*, *geninterrupt*, *intdos*, *intdosx*, *int86*, *intr*, *segread*

Beispiel

```
#include <dos.h>
#include <process.h>
#include <stdio.h>

int main(void)
{
   char filename[80];
   union REGS inregs, outregs;
   struct SREGS segregs;

   printf("Enter filename: ");
   gets(filename);
   inregs.h.ah = 0x43;
   inregs.h.al = 0x21;
   inregs.x.dx = FP_OFF(filename);
   segregs.ds = FP_SEG(filename);
   int86x(0x21, &inregs, &outregs, &segregs);
   printf("File attribute: %X\n", outregs.x.cx);
   return 0;
}
```

intdos

Funktion
Allgemeine Schnittstelle für DOS-Interrupts.

Syntax
```
#include <dos.h>
int intdos(union REGS *inregs, union REGS *outregs);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung
intdos führt einen Aufruf des DOS-Interrupt 0x21 aus, wobei die Nummer der auszuführenden Funktion durch *inregs->h.ah* angegeben sein muß.

Nach der Rückkehr vom Interrupt 0x21 kopiert *intdos* die Registerwerte in *outregs*, den Status des Carry-Flag in das Feld *x.cflag* von *outregs* und den Inhalt des 8086-Flag-Register in das Feld *x.flags* von *outregs*. Ein gesetztes Carry-Flag zeigt einen Fehler an.

Hinweis: inregs und *outregs* können auf dieselbe Struktur zeigen.

Rückgabewert
intdos liefert den von DOS gesetzten Inhalt des Registers AX zurück. Wenn das Carry-Flag durch den Aufruf gesetzt wurde (*outregs->x.cflag != 0*), dann wird der globalen Variablen *_doserrno* der entsprechende Fehlercode zugewiesen.

Siehe auch
bdos, bdosptr, geninterrupt, int86, int86x, intdosx, intr

Beispiel
```
#include <stdio.h>
#include <dos.h>

/* deletes file name; returns 0 on success,
   nonzero on failure
*/
int delete_file(char near *filename)
{
   union REGS regs;
   int ret;
   regs.h.ah = 0x41;
/* delete file */
   regs.x.dx = (unsigned) filename;
   ret = intdos(&regs, &regs);

   /* if carry flag is set, there was an error */
   return(regs.x.cflag ? ret : 0);
}

int main(void)
{
   int err;
   err = delete_file("NOTEXIST.$$$");
```

```
    if (!err)
      printf("Able to delete NOTEXIST.$$$\n");
    else
      printf("Not able to delete NOTEXIST.$$$\n");
    return 0;
}
```

intdosx

Funktion — Allgemeine Schnittstelle für DOS-Interrupt.

Syntax

```
#include <dos.h>
int intdosx(union REGS *inregs, union REGS *outregs,
            struct SREGS *segregs);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *intdosx* führt einen Aufruf des DOS-Interrupt 0x21 aus, wobei die Nummer der auszuführenden Funktion durch *inregs->h.ah* angegeben sein muß.

Zusätzlich kopiert *intdosx* die Werte von *segregs->ds* und *segregs->es* in die entsprechenden Register, bevor es den Interrupt ausführt. Auf diese Weise kann ein Programm, das **far** Zeiger oder ein großes Speichermodell benutzt, angeben, welches Segment für den Software-Interrupt verwendet werden soll.

Nach der Rückkehr vom Interrupt 0x21 kopiert *intdosx* die Registerwerte in *outregs*, den Status des Carry-Flag in das Feld *x.cflag* von *outregs* und das 8086-Flag-Register in das Feld *x.flags* von *outregs*. Anschließend stellt *int86x* den alten Inhalt des DS-Registers wieder her und speichert die Werte der Segmentregister in den Feldern *segregs->es* bzw. *segregs->ds*. Ist das Carry-Flag gesetzt, zeigt das im allgemeinen einen Fehler an.

intdosx ermöglicht den Aufruf eines 8086-Software-Interrupt, der mit einem anderen Wert von DS als dem voreingestellten arbeitet und/oder ein Argument in ES benötigt.

Hinweis: inregs und *outregs* können auf dieselbe Struktur zeigen.

Rückgabewert — *intdosx* liefert den von DOS gesetzten Inhalt des Registers AX zurück. Wenn das Carry-Flag durch den Aufruf gesetzt wurde (*outregs->x.cflag !=0*), wird der globalen Variablen *_doserrno* der entsprechende Fehlercode zugewiesen.

Siehe auch — *bdos*, *bdosptr*, *geninterrupt*, *int86*, *int86x*, *intdos*, *intr*, *segread*

Beispiel

```
#include <stdio.h>
#include <dos.h>
```

```
/* deletes file name; returns 0 on success,
   nonzero on failure */
int delete_file(char far *filename)
{
   union REGS regs; struct SREGS sregs;
   int ret;
   regs.h.ah = 0x41;                /* delete file */
   regs.x.dx = FP_OFF(filename);
   sregs.ds = FP_SEG(filename);
   ret = intdosx(&regs, &regs, &sregs);

   /* if carry flag is set, there was an error */
   return(regs.x.cflag ? ret : 0);
}

int main(void)
{
   int err;
   err = delete_file("NOTEXIST.$$$");
   if (!err)
      printf("Able to delete NOTEXIST.$$$\n");
   else
      printf("Not Able to delete NOTEXIST.$$$\n");
   return 0;
}
```

intr

Funktion — Alternative 8086-Software-Interrupt-Schnittstelle.

Syntax

```
#include <dos.h>
void intr(int intno, struct REGPACK *preg);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *intr* führt den durch *intno* angegebenen Software-Interrupt aus und verwendet die REGPACK-Struktur **preg* zur Übergabe von Prozessor-Registern bzw. Ergebnissen.

Vor der Ausführung des Interrupts werden die Prozessor-Register mit den Werten der Strukturelemente gesetzt, nach der Ausführung folgt eine Kopie der Register (und der Flags) in die Struktur **preg*.

Das Argument *intno* ist die Nummer des Interrupts und *preg* die Adresse einer Struktur, die a) die Werte enthält, die vor dem Interrupt-Aufruf in den Registern gespeichert werden, und b) die Werte der Register nach dem Interrupt-Aufruf enthält.

Der Strukturtyp REGPACK ist in dos.h folgendermaßen definiert:

```
struct REGPACK {
   unsigned r_ax, r_bx, r_cx, r_dx;
   unsigned r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
```

Rückgabewert Keiner. Die REGPACK-Struktur **preg* enthält die Registerwerte nach dem Aufruf des Interrupts.

Siehe auch *geninterrupt, int86, int86x, intdos, intdosx*

Beispiel

```
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <dos.h>

#define CF 1  /* Carry flag */

int main(void)
{
   char directory[80];
   struct REGPACK reg;

   printf("Enter directory to change to: ");
   gets(directory);
   reg.r_ax = 0x3B << 8;    /* shift 3Bh into  AH */
   reg.r_dx = FP_OFF(directory);
   reg.r_ds = FP_SEG(directory);
   intr(0x21, &reg);
   if (reg.r_flags & CF)
      printf("Directory change failed\n");
   getcwd(directory, 80);
   printf("The current directory is: %s\n",directory);
   return 0;
}
```

ioctl

Funktion Direkte Steuerung von Peripheriegeräten.

Syntax

```
#include <io.h>
int ioctl(int handle, int func
          [, void *argdx, int argcx]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *ioctl* ist im Prinzip auf UNIX-Systemen verfügbar – sowohl die erwarteten Parameter als auch die ausgeführten Aktionen sind allerdings anders definiert. Selbst innerhalb von UNIX arbeitet diese Funktion nicht einheitlich – System III und die Version 7 implementieren sie auf verschiedene Weise. Die DOS-Funk-

tion *ioctl* ist deshalb nicht nach UNIX portierbar (und nur eingeschränkt portierbar zwischen verschiedenen DOS-Versionen). Unter der DOS-Version 3.0 sind die Werte 8 und 11 für *func* definiert.

ioctl verwendet die DOS-Funktion 0x44, wobei die tatsächlich ausgeführten Aktionen sehr stark von *func* abhängen.

Die folgenden Werte sind für *func* definiert:

0 Lesen von Informationen über das Gerät.

1 Setzen von Geräteinformationen durch den Parameter *argdx*.

2 Lesen von *argcx* Bytes vom Gerät in den Bereich, auf den *argdx* zeigt.

3 Schreiben von *argcx* Bytes zu dem Gerät, *argdx* zeigt auf die Startadresse im Speicher.

4 Wie Subfunktion 2, allerdings wird der Wert von *handle* hier als Laufwerksnummer interpretiert (0 =Standard, 1 =A, 2 =B, etc.).

5 Wie Subfunktion 3, allerdings mit *handle* als Laufwerksnummer (0 =Standard, 1 =A, 2 =B usw.).

6 Ermittlung des Eingabestatus.

7 Ermittlung des Ausgabestatus.

8 Prüfung, ob der Datenträger auswechselbar ist (nur DOS-Version 3.0).

11 Setzen der Anzahl der Wiederholversuche bei Kollisionen im SHARED-Betrieb (nur DOS-Version 3.0).

ioctl wird normalerweise für Geräteeinheiten (»Kommunikationskanäle«) verwendet, kann aber auch zusammen mit regulären Diskettendateien eingesetzt werden. Im letzteren Fall sind allerdings nur die Subfunktionen 0, 6 und 7 definiert – alle anderen Aufrufe liefern einen Fehler zurück und setzen *errno* auf den Wert EINVAL. Genaue Informationen zur DOS-Funktion 0x44 (IOCTL), deren Parameter und Rückgabewerte finden Sie in ihrem DOS-Referenzhandbuch.

Die Argumente *argdx* und *argcx* sind in verschiedenen Fällen optional. *ioctl* stellt eine direkte Schnittstelle zu speziellen Funktionen von DOS-Gerätetreibern dar.

Das exakte Verhalten dieser Funktion hängt daher stark von der verwendeten Hardware und den angesprochenen Geräten ab. Nicht alle Hardware-Hersteller halten sich an die Schnittstelle, wie sie hier beschrieben wurde. Für die korrekte Benutzung von *ioctl* benötigen Sie die BIOS-Dokumentation des Herstellers.

Rückgabewert Für die Werte 0 und 1 von *func* liefert *ioctl* den Inhalt des Prozessor-Registers DX zurück – die gelesene/gesetzte Geräte-Information.

Für Werte von *func* im Bereich von 2 bis 5 wird die Anzahl der tatsächlich gelesenen/geschriebenen Bytes zurückgeliefert.

Für die Werte 6 und 7 wird die Statusinformation zurückgeliefert.

Im Falle eines Fehlers liefert *ioctl* den Wert -1 und setzt *errno* auf einen der folgenden Werte:

EINVAL	Nicht definierte/anwendbare Subfunktion
EBADF	Ungültiges Handle
EINVDAT	Ungültige Daten

Beispiel

```
#include <stdio.h>
#include <dir.h>
#include <io.h>

int main(void)
{
   int stat;

   /* use func 8 to determine if the
      default drive is removable
   */
   stat = ioctl(0, 8, 0, 0);
   if (!stat)
      printf("Drive %c is removable.\n",
             getdisk() + 'A');
   else
      printf("Drive %c is not removable.\n",
             getdisk() + 'A');
   return 0;
}
```

isalnum

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int isalnum(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isalnum* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*). Das Makro liefert

nur dann sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isalnum* liefert einen Wert ungleich Null zurück, wenn *c* ein Buchstabe (A bis Z, a bis z) oder eine Ziffer (0 bis 9) ist.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isalnum(c))
      printf("%c is alphanumeric\n",c);
   else
      printf("%c isn't alphanumeric\n",c);
   return 0;
}
```

isalpha

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int isalpha(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isalpha* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Dieses Makro liefert nur dann sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isalpha* liefert einen Wert ungleich Null zurück, wenn *c* ein Buchstabe (A bis Z, a bis z) ist.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isalpha(c))
      printf("%  c is alphabetic\n",c);
```

```
   else
      printf("%c isn't alphabetic\n",c);

   return 0;
}
```

isascii

Funktion Makro zur Klassifizierung von Zeichen.

Syntax
```
#include <ctype.h>
int isascii(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *isascii* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

isascii ist für alle Integerwerte definiert.

Rückgabewert *isascii* liefert einen Wert ungleich Null zurück, wenn das niederwertige Byte von *c* im Bereich von 0 bis 127 (0x00-0x7F) liegt.

Beispiel
```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isascii(c))
      printf("%c is ascii\n",c);
   else
      printf("%c isn't ascii\n",c);

   return 0;
}
```

isatty

Funktion Prüft den Gerätetyp.

Syntax
```
#include <io.h>
int isatty(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *isatty* prüft, ob das über *handle* angegebene Peripheriegerät in eine der folgenden Klassen fällt:

- Terminal
- Tastatur
- Drucker
- serielle Schnittstelle

Rückgabewert Wenn das Gerät zeichenorientiert arbeitet, liefert *isatty* einen Wert ungleich Null zurück, ansonsten den Wert 0.

Beispiel

```
#include <stdio.h>
#include <io.h>

int main(void)
{
   int handle;

   handle = fileno(stdprn);
   if (isatty(handle))
      printf("Handle %d is a device type\n", handle);
   else
      printf("Handle %d isn't a device type\n",
             handle);
   return 0;
}
```

iscntrl

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int iscntrl(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *iscntrl* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *iscntrl* liefert einen Wert ungleich Null zurück, wenn ein Steuerzeichen (0x00 bis 0x1F) oder das Zeichen DEZ (0x7F) gefunden wurde.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (iscntrl(c))
      printf("%c is a control character\n",c);
   else
      printf("%c isn't a control character\n",c);
   return 0;
}
```

isdigit

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int isdigit(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isdigit* prüft das übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isdigit* liefert einen Wert ungleich Null zurück, wenn *c* eine Ziffer (0 bis 9) ist.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isdigit(c))
      printf("%c is a digit\n",c);
   else
      printf("%c isn't a digit\n",c);
   return 0;
}
```

isgraph

Funktion Makro zur Klassifizierung von Zeichen.

Syntax
```
#include <ctype.h>
int isgraph(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isgraph* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isgraph* liefert einen Wert ungleich Null zurück, wenn *c* ein druckbares Zeichen ist. *isgraph* ähnelt dem Makro *isprint*, schließt aber das Leerzeichen aus.

Beispiel
```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isgraph(c))
      printf("%c is a graphic character\n",c);
   else
      printf("%c isn't a graphic character\n",c);

   return 0;
}
```

islower

Funktion Makro zur Klassifizierung von Zeichen.

Syntax
```
#include <ctype.h>
int islower(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *islower* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert

einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *islower* liefert einen Wert ungleich Null zurück, wenn *c* ein Kleinbuchstabe (a bis z) ist.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (islower(c))
      printf("%c is lower case\n",c);
   else
      printf("%c isn't lower case\n",c);
   return 0;
}
```

isprint

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int isprint(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isprint* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isprint* liefert einen Wert ungleich Null zurück, wenn *c* ein druckbares Zeichen (0x20 bis 0x7E) ist.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
```

```
   if (isprint(c))
      printf("%c is a printable character\n",c);
   else
      printf("%c isn't a printable character\n",c);

   return 0;
}
```

ispunct

Funktion Makro zur Klassifizierung von Zeichen.

Syntax
```
#include <ctype.h>
int ispunct(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *ispunct* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *ispunct* liefert einen Wert ungleich Null zurück, wenn *c* ein Interpunktionszeichen ist (*iscntrl* oder *isspace*).

Beispiel
```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (ispunct(c))
      printf("%c is a punctuation character\n",c);
   else
      printf("%c isn't a punctuation character\n",c);
   return 0;
}
```

isspace

Funktion Makro zur Klassifizierung von Zeichen.

Syntax
```
#include <ctype.h>
int isspace(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isspace* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isspace* liefert einen Wert ungleich Null zurück, wenn *c* ein Zeilenvorschub, Tabulator, Leerzeichen, vertikaler Tabulator oder ein Seitenvorschub ist (ASCII-Codes 0x09 bis 0x0D, 0x20).

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isspace(c))
      printf("%c is white space\n",c);
   else
      printf("%c isn't white space\n",c);
   return 0;
}
```

isupper

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int isupper(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isupper* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit #*undef* als Funktion verfügbar machen.

Rückgabewert *isupper* liefert einen Wert ungleich Null zurück, wenn *c* ein Großbuchstabe (A bis Z) ist.

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isupper(c))
      printf("%  c is upper case\n",c);
   else
      printf("%c isn't upper case\n",c);
   return 0;
}
```

isxdigit

Funktion Makro zur Klassifizierung von Zeichen.

Syntax

```
#include <ctype.h>
int isxdigit(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *isxdigit* prüft das als Integer übergebene ASCII-Zeichen anhand einer Tabelle auf verschiedene Bedingungen hin und liefert einen Wert ungleich Null (*true*) zurück, wenn die Bedingung zutrifft, ansonsten den Wert 0 (*false*).

Das Makro liefert nur sinnvolle Resultate, wenn *isascii*(*c*) *true* liefert oder wenn *c* den Wert EOF (-1) darstellt.

Sie können das Makro mit *#undef* als Funktion verfügbar machen.

Rückgabewert *isxdigit* liefert einen Wert ungleich Null zurück, wenn *c* eine Hex-Ziffer ist (0 bis 9, A bis F, a bis f).

Beispiel

```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
   char c = 'C';
   if (isxdigit(c))
      printf("%c is hexadecimal\n",c);
   else
      printf("%c isn't hexadecimal\n",c);
   return 0;
}
```

itoa

Funktion — Konvertiert einen Integerwert in einen String.

Syntax

```
#include <stdlib.h>
char *itoa(int value, char *string, int radix);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung — *itoa* konvertiert den übergebenen Wert *value* in einen nullterminierten String und speichert das Ergebnis in der durch *string* bezeichneten Variablen. *itoa* erwartet einen Integerwert als *value*.

Der Parameter *radix* legt die Basis fest, auf der die Konvertierung stattfinden soll – mögliche Werte liegen im Bereich von 2 bis 36.

Wenn *radix* den Wert 10 hat und *value* kleiner als Null ist, stellt *itoa* dem Ergebnis ein Minuszeichen voran.

Hinweis: Der übergebene *string* muß lang genug sein, um das Ergebnis der Konvertierung (inklusive eines abschließenden Nullzeichens »\0«) aufnehmen zu können. *itoa* erzeugt maximal 17 Bytes.

Rückgabewert — *itoa* liefert einen Zeiger auf *string* zurück.

Siehe auch — *ltoa, utoa*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   int number = 12345;
   char string[25];

   itoa(number, string, 10);
   printf("integer = %d string = %s\n",number,string);
   return 0;
}
```

kbhit

Funktion — Prüft auf verfügbare Tastenanschläge.

Syntax

```
#include <conio.h>
int kbhit(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *kbhit* prüft, ob ein Zeichen über die Tastatur eingegeben wurde. Verfügbare Zeichen können mit *getch* oder *getche* gelesen werden

Rückgabewert Wenn eine noch nicht gelesene Eingabe vorliegt, liefert *kbhit* einen Wert ungleich Null. Liegt kein Zeichen an, dann liefert *kbhit* den Wert 0.

Siehe auch *getch*, *getche*

Beispiel

```
#include <conio.h>

int main(void)
{
   cprintf("Press any key to continue:");
   while (!kbhit()) /* do nothing */ ;
   cprintf("\r\nA key was pressed...\r\n");
   return 0;
}
```

keep, _dos_keep

Funktion Beendet ein Programm und macht es speicherresident.

Syntax

```
#include <dos.h>
void keep(unsigned char status, unsigned size);
void _dos_keep(unsigned char status, unsigned size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *keep* und *_dos_keep* verwenden die DOS-Funktion 0x31, um ein laufendes Programm zu beenden und den durch *status* angegebenen Wert als Exit-Code zurückzuliefern. Das beendete Programm bleibt resident und wird auf die über *size* angegebene Anzahl von Paragraphen (Einheiten zu je 16 Byte) gesetzt, der Rest des vom Programm belegten Speichers wird freigegeben.

Verwenden Sie *keep* und *_dos_keep* bei der Installation von TSR-Programmen.

Bevor *_dos_keep* das laufende Programm beendet, werden alle mit *atexit* festgelegten Exit-Funktionen aufgerufen, Dateipuffer entleert und durch den Startup-Code modifizierte Interrupt-Vektoren wiederhergestellt.

Rückgabewert Keiner.

Siehe auch *abort*, *exit*

Beispiel

```
/* NOTE:
  This is an interrupt service routine.
```

```
  You can NOT compile this program with Test
  Stack Overflow turned on and get an
  executable file which will operate
  correctly.  Due to the nature of this
  function the formula used to compute
  the number of paragraphs may not
  necessarily work in all cases.  Use with
  care!  Terminate Stay Resident (TSR)
  programs are complex and no other support
  for them is provided.  Refer to the
  MS-DOS technical documentation
  for more information.  *

#include <dos.h>
/* The clock tick interrupt */
#define INTR 0x1C
/* Screen attribute (blue on grey) */
#define ATTR 0x7900

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

/* reduce heaplength and stacklength to make
   a smaller program in memory

*/
extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 512;

void interrupt ( *oldhandler)(__CPPARGS);

typedef unsigned int (far *s_arrayptr);

void interrupt handler(__CPPARGS)
{
   s_arrayptr screen[80];
   static int count;

/* For a color screen the video memory is at
   B800:0000. For a monochrome system use B000:000
*/
   screen[0] = (s_arrayptr) MK_FP(0xB800,0);

/* increase the counter and keep it within 0 to 9 */
   count++;
   count %= 10;

/* put the number on the screen */
   screen[0][79] = count + '0' + ATTR;

/* call the old interrupt handler */
```

```
   oldhandler();
}

int main(void)
{

/* get the address of the current clock
   tick interrupt */
oldhandler = getvect(INTR);

/* install the new interrupt handler */
setvect(INTR, handler);

/* _psp is the starting address of the
   program in memory.  The top of the stack
   is the end of the program.  Using _SS and
   _SP together we can get the end of the
   stack.  You may want to allow a bit of
   saftey space to insure that enough room
   is being allocated ie:
   (_SS + ((_SP + safety space)/16) - _psp)
*/
keep(0, (_SS + (_SP/16) - _psp));
return 0;
}
```

Beispiel 2

```
/* NOTE:
  This is an interrupt service routine.
  You can NOT compile this program with Test
  Stack Overflow turned on and get an
  executable file which will operate
  correctly.  Due to the nature of this
  function the formula used to compute
  the number of paragraphs may not
  necessarily work in all cases.  Use with
  care!  Terminate Stay Resident (TSR)
  programs are complex and no other support
  for them is provided.  Refer to the
  MS-DOS technical documentation
  for more information.  */

#include <dos.h>
/* The clock tick interrupt */
#define INTR 0x1C
/* Screen attribute (blue on grey) */
#define ATTR 0x7900

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif
```

```
/* reduce heaplength and stacklength to make
   a smaller program in memory
*/
extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 512;

void interrupt ( *oldhandler)(__CPPARGS);

typedef unsigned int (far *s_arrayptr);

void interrupt handler(__CPPARGS)
{
  s_arrayptr screen[80];
  static int count;

/* For a color screen the video memory is at
   B800:0000. For a monochrome system use B000:000
*/
   screen[0] = (s_arrayptr) MK_FP(0xB800,0);

/* increase the counter and keep it within 0 to 9 */
   count++;
   count %= 10;

/* put the number on the screen */
   screen[0][79] = count + '0' + ATTR;

/* call the old interrupt handler */
   oldhandler();
}

int main(void)
{

/* get the address of the current clock
   tick interrupt */
oldhandler = _dos_getvect(INTR);

/* install the new interrupt handler */
_dos_setvect(INTR, handler);

/* _psp is the starting address of the
   program in memory.  The top of the stack
   is the end of the program.  Using _SS and
   _SP together we can get the end of the
   stack.  You may want to allow a bit of
   saftey space to insure that enough room
   is being allocated ie:
   (_SS + ((_SP + safety space)/16) - _psp)
*/
_dos_keep(0, (_SS + (_SP/16) - _psp));
return 0;
}
```

labs

Funktion Liefert den absoluten Wert eines **long**-Wertes.

Syntax
```
#include <math.h>
long int labs(long int x);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *labs* berechnet den absoluten Wert des **long**-Arguments *x*.

Rückgabewert *labs* liefert den absoluten Wert seines **long**-Arguments *x* zurück.

Siehe auch *abs*, *caps*, *fabs*

Beispiel
```
#include <stdio.h>
#include <math.h>

int main(void)
{
   long result;
   long x = -12345678L;

   result= labs(x);
   printf("number: %ld abs value: %ld\n", x, result);

   return 0;
}
```

ldexp, ldexpl

Funktion Berechnet $x * 2^{exp}$.

Syntax
```
#include <math.h>
double ldexp(double x, int exp);
long double ldexpl(long double x, int exp);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*ldexp*
x		x			*ldexpl*

Beschreibung *ldexp* liefert das Ergebnis der Berechnung $x * 2^{exp}$ als **double** Wert zurück.

ldexpl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert Bei fehlerfreier Ausführung liefern *ldexp* und *ldexpl* das Ergebnis der Berechnung von $x * 2^{exp}$ zurück.

Die Behandlung von Fehlern kann für diese Funktionen über die Routine *matherr* bzw. *_matherrl* modifiziert werden.

Siehe auch *exp, frexp, modf*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double value;
   double x = 2;

   /* ldexp raises 2 by a power of 3
      then multiplies the result by 2   */
   value = ldexp(x,3);
   printf("The ldexp value is: %lf\n", value);

   return 0;
}
```

ldiv

Funktion Dividiert zwei **long**-Werte und liefert den Quotienten und den Divisionsrest zurück.

Syntax

```
#include <stdlib.h>
ldiv_t ldiv(long int numer, long int denom)
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *ldiv* dividiert die als *numer* (Dividend) und *denom* (Divisor) übergebenen **long**-Werte und liefert eine Struktur des Typs *ldiv_t* zurück. Dieser Datentyp ist in stdlib.h folgendermaßen definiert:

```
typedef struct {
   long int quot;      /* Quotient */
   long int rem;       /* Divisionsrest */
} ldiv_t;
```

Rückgabewert Es wird eine Struktur des zuvor beschriebenen Typs zurückgeliefert.

Siehe auch *div*

Beispiel

```
/* ldiv example */

#include <stdlib.h>
#include <stdio.h>

int main(void)
```

```
{
   ldiv_t lx;

   lx = ldiv(100000L, 30000L);
   printf("100000 div 30000 = %ld remainder %ld\n",
          lx.quot, lx.rem);
   return 0;
}
```

lfind

Funktion — Führt eine lineare Suche nach einem Array-Element durch.

Syntax

```
#include <stdlib.h>
void *lfind(const void *key, const void *base,
            size_t *num, size_t width,
            int (*fcmp)(const void*, const void *));
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *lfind* durchsucht ein sequentiell angeordnetes Array nach dem Wert des Schlüsselfeldes (*key*). Die Suche erfolgt linear. *lfind* verwendet eine vom Programmierer erstellte Vergleichsroutine (*fcmp*)

Der Parameter *base* muß auf das erste Element des Array zeigen, *num* auf einen Integer, der die Anzahl der Array-Elemente enthält. *width* gibt die Größe eines einzelnen Elements in Bytes an.

Rückgabewert — *lfind* liefert einen Zeiger auf das erste Element des Array zurück, das mit dem gesuchten Schlüsselwort identisch ist. Falls der gesuchte Eintrag nicht existiert, liefert *lfind* den Wert NULL. Die Vergleichsroutine muß für **elem1* == **elem2* (das sind ihre beiden Parameter) den Wert NULL liefern, andernfalls einen Wert ungleich NULL.

Siehe auch — *bsearch, lsearch, qsort*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
int compare(int *x, int *y)
{
   return( *x - *y );
}
int main(void)
{
   int array[5] = {35, 87, 46, 99, 12};
   size_t nelem = 5;
```

```
   int key;
   int *result;
   key = 99;
   result = (int *) lfind(&key, array, &nelem,
             sizeof(int), (int(*)(const void *,
                 const void *))compare);
   if (result)
      printf("Number %d found\n",key);
   else
      printf("Number %d not found\n",key);
   return 0;
}
```

line

Funktion — Zeichnet eine Linie zwischen zwei Punkten.

Syntax

```
#include <graphics.h>
void far line(int x1, int y1, int x2, int y2);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Die Funktion verwendet die mit *setlinestyle* gesetzte Linienart und -dicke sowie die aktuell gesetzte Farbe.

line zeichnet eine Linie vom Punkt (*x1,y1*) zum Punkt (*x2,y2*), ohne die aktuelle Position (CP) dabei zu verändern.

Rückgabewert — Keiner.

Siehe auch — *getlinesettings, linerel, lineto, setcolor, setlinestyle, setwritemode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
```

```
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   setcolor(getmaxcolor());
   xmax = getmaxx();
   ymax = getmaxy();

   /* draw a diagonal line */
   line(0, 0, xmax, ymax);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

linerel

Funktion — Zeichnet eine Linie relativ zur aktuellen Position (CP).

Syntax

```
#include <graphics.h>
void far linerel(int dx, int dy);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *linerel* interpretiert die als *dx,dy* übergebenen Werte als relative Entfernung zur aktuellen Position (CP) und zeichnet eine Linie von der aktuellen Position zum durch (*dx,dy*) gegebenen Punkt. Die aktuelle Position wird dadurch um (*dx,dy*) bewegt.

Rückgabewert — Keiner.

Siehe auch — *getlinesettings, line, lineto, setcolor, setlinestyle, setwritemode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];
```

```
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   /* move the C.P. to location (20, 30) */
   moveto(20, 30);

   /* create and output a
      message at (20, 30) */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtextxy(20, 30, msg);

   /* draw a line to a point a relative
      distance away from the current
      value of C.P.   */
   linerel(100, 100);

   /* create and output a message at C.P. */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

lineto

Funktion

Zeichnet eine Linie von der aktuellen Position (CP) zum angegebenen Punkt.

Syntax

```
#include <graphics.h>
void far lineto(int x, int y);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

lineto zeichnet eine Linie von der aktuellen Position (CP) zum Punkt (x,y). Die aktuelle Position ist danach der Punkt (x,y).

Rückgabewert Keiner.

Siehe auch *getlinesettings, line, linerel, setcolor, setlinestyle, setvisualpage, setwritemode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   /* move the C.P. to location (20, 30) */
   moveto(20, 30);

   /* create and output a
      message at (20, 30) */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtextxy(20, 30, msg);

   /* draw a line to (100, 100) */
   lineto(100, 100);

   /* create and output a message at C.P. */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

localeconv

Funktion Liefert einen Zeiger auf die aktuelle lokale Struktur.

Syntax
```
#include <locale.h>
struct lconv *localeconv(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *localeconv* setzt länderspezifische numerische Formate (z.B. für die Währung). Borland C++ unterstützt die *locale*-Variable C.

Rückgabewert *localeconv* liefert einen Zeiger auf die aktuelle lokale Struktur zurück (Details finden Sie in locale.h).

Siehe auch *setlocale*

Beispiel
```
#include <locale.h>
#include <stdio.h>

int main(void)
{
   struct lconv ll;
   struct lconv *conv = &ll;

/* read the locality conversion structure */
   conv = localeconv();

/* display the structure */
   printf("Decimal Point                  : "
          "%s\n", conv->decimal_point);
   printf("Thousands Separator            : "
          "%s\n", conv->thousands_sep);
   printf("Grouping                       : "
          "%s\n", conv->grouping);
   printf("International Currency symbol  : "
          "%s\n", conv->int_curr_symbol);
   printf("$ thousands separator          : "
          "%s\n", conv->mon_thousands_sep);
   printf("$ grouping                     : "
          "%s\n", conv->mon_grouping);
   printf("Positive sign                  : "
          "%s\n", conv->positive_sign);
   printf("Negative sign                  : "
          "%s\n", conv->negative_sign);
   printf("International fraction digits  : "
          "%d\n", conv->int_frac_digits);
   printf("Fraction digits                : "
          "% d\n", conv->frac_digits);
```

```
   printf("Positive $ symbol precedes     : "
          "%d\n", conv->p_cs_precedes);
   printf("Positive sign space separation : "
          "%d\n", conv->p_sep_by_space);
   printf("Negative $ symbol precedes     : "
          "%d\n", conv->n_cs_precedes);
   printf("Negative sign space separation : "
          %d\n", conv->n_sep_by_space);
   printf("Positive sign position         : "
          "%d\n", conv->p_sign_posn);
   printf("Negative sign position         : "
          "%d\n", conv->n_sign_posn);
   return 0;
}
```

localtime

Funktion

Konvertiert Datum und Uhrzeit in eine Struktur des Typs *tm*.

Syntax

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung

localtime übernimmt die Adresse eines Werts, der durch *time* geliefert wird, und gibt einen Zeiger auf die Struktur des Typs *tm* zurück, die die umgerechneten Zeitangaben enthält. *localtime* berücksichtigt hierbei Zeitzonen und die Sommerzeit.

Die globale **long**-Variable *timezone* muß auf den Unterschied zwischen der lokalen Zeit und der *Greenwich Mean Time* (GMT) in Sekunden gesetzt werden. Die globale Variable *daylight* darf nur dann auf einen Wert ungleich Null gesetzt werden, wenn die Standardwerte der USA für Sommer- und Winterzeit angewendet werden können.

Der Strukturtyp *tm* ist in time.h folgendermaßen definiert:

```
struct tm {
   int tm_sec;     /* Sekunden */
   int tm_min;     /* Minuten */
   int tm_hour;    /* Stunden (0..23) */
   int tm_mday;    /* Tag im Monat (1..31) */
   int tm_mon;     /* Monat (0..11) */
   int tm_year;    /* Jahr (-1900) */
   int tm_wday;    /* Wochentag (0..6) */
   int tm_yday;    /* Tag im Jahr (0..365) */
   int tm_isdst;   /* Sommerzeit */
};
```

Die Zählung der Wochentage beginnt mit Sonntag (Wert 0). Wenn das Flag *tm_isdst* einen Wert ungleich Null hat, wird mit Sommerzeit gerechnet.

Rückgabewert *localtime* liefert einen Zeiger auf eine statische Struktur, die die umgerechneten Zeitangaben enthält. Die Struktur wird durch jeden Aufruf dieser Funktionen erneut gesetzt.

Siehe auch *asctime*, *ctime*, *ftime*, *gmtime*, *stime*, *time*, *tzset*

Beispiel

```
#include <time.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   time_t timer;
   struct tm *tblock;

   /* gets time of day */
   timer = time(NULL);

   /* converts date/time to a structure */
   tblock = localtime(&timer);
   printf("Local time is: %s", asctime(tblock));

   return 0;
}
```

lock

Funktion Sperrt Teile einer Datei im Modus SHARED gegen Zugriffe anderer Programme.

Syntax

```
#include <io.h>
int lock(int handle, long offset, long length);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Ab der Version 3.x von DOS ist der Modus SHARED definiert, in dem eine Datei simultan von mehreren Programmen genutzt werden kann. Bevor *lock* verwendet wird, muß die Datei SHARE.EXE geladen werden.

lock sperrt *length* Bytes der durch *handle* bezeichneten Datei ab der durch *offset* vorgegebenen Position gegen Zugriffe anderer Programme. Mit *lock* lassen sich beliebig viele Bereiche einer Datei sperren – überlappen dürfen sich diese Bereiche allerdings nicht. Jedes Programm, das auf einen gesperrten Bereich zugreifen will, wiederholt diese Operation dreimal und bricht dann mit einem Fehler ab.

Rückgabewert Die Funktion liefert bei fehlerfreier Ausführung den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und die globale Variable *errno* auf folgenden Wert gesetzt:

EACCES	Zugriff nicht gestattet

Siehe auch *open, sopen, unlock*

Beispiel

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
   int handle, status;
   long length;

   /* Must have DOS Share.exe loaded for */
   /* file locking to function properly */

   handle = sopen("c:\\autoexec.bat",
      O_RDONLY,SH_DENYNO,S_IREAD);

   if (handle < 0)
   {
      printf("sopen failed\n");
      exit(1);
   }

   length = filelength(handle);
   status = lock(handle,0L,length/2);

   if (status == 0)
      printf("lock succeeded\n");
   else
      printf("lock failed\n");

   status = unlock(handle,0L,length/2);

   if (status == 0)
      printf("unlock succeeded\n");
   else
      printf("unlock failed\n");

   close(handle);
   return 0;
}
```

locking

Funktion

Sperrt oder entsperrt Teile einer Datei im Modus SHARED.

Syntax

```
#include <io.h>
#include <sys\locking.h>
int locking(int handle, int cmd, long length);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

Ab der Version 3.x von DOS ist der Modus SHARED definiert, in dem eine Datei simultan von mehreren Programmen genutzt werden kann. Bevor *locking* verwendet wird, muß die Datei SHARE.EXE geladen werden.

locking sperrt *length* Bytes der durch *handle* bezeichneten Datei ab der durch *offset* vorgegebenen Position gegen Zugriffe anderer Programme.

Mit *locking* lassen sich beliebig viele Bereiche einer Datei sperren – überlappen dürfen sich diese Bereiche allerdings nicht. Jedes Programm, das auf einen gesperrten Bereich zugreifen will, wiederholt diese Operation dreimal und bricht dann mit einem Fehler ab.

cmd gibt die durchzuführende Operation an. Folgende Konstanten sind dafür in sys\locking.h definiert:

LK_LOCK	Sperren des Bereichs. Ist das Sperren nicht möglich, wird der Sperrversuch 10 Sekunden lang im 1-Sekunden-Takt wiederholt. Anschließend Abbruch.
LK_RLCK	Wie LK_LOCK.
LK_NBLCK	Sperren des Bereichs. Sofortiger Abbruch, wenn Sperre nicht möglich ist.
LK_NBRLCK	Wie LK_NBLCK.
LK_UNLCK	Entsperren des Bereichs (der Bereich muß vorher gesperrt worden sein).

Rückgabewert

Die Funktion liefert bei fehlerfreier Ausführung den Wert 0. Andernfalls wird der Wert -1 zurückgeliefert und die globale Variable *errno* auf einen der folgenden Werte gesetzt:

EBADF	Ungültiges Dateihandle
EACCES	Datei bereits gesperrt bzw. entsperrt
EDEADLOCK	Datei konnte nach 10 Versuchen nicht gesperrt werden (*cmd* ist LK_LOCK oder LK_RLCK)
EINVAL	Ungültiges *cmd* oder SHARE.EXE nicht geladen

Siehe auch *_fsopen, open, sopen*

Beispiel

```
#include <io.h>
#include <fcntl.h>
#include <process.h>
#include <share.h>
#include <stdio.h>
#include <sys\locking.h>

int main(void)
{
  int handle, status;
  long length;

  /* must have DOS SHARE.EXE loaded
     for file locking to function
  */
  handle = sopen("c:\\autoexec.bat",
                 O_RDONLY,SH_DENYNO);
  if (handle < 0) {
    printf("sopen failed\n");
    exit(1);
  }
  length = filelength(handle);
  status = locking(handle,LK_LOCK,length/2);
  if (status == 0)
    printf("lock succeeded\n");
  else
    perror("lock failed");
  status = locking(handle,LK_UNLCK,length/2);
  if (status == 0)
    printf("unlock succeeded\n");
  else
    perror("unlock failed");
  close(handle);
  return 0;
}
```

log, logl

Funktion Berechnet den natürlichen Logarithmus des Arguments.

Syntax

reelle Version:

```
#include <math.h>
double log(double x);
long double logl(long double x);
```

komplexe Version:

```
#include <complex.h>
complex log(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*logl*
x	x	x	x		*reelles log*
x		x		x	*complexes log*

Beschreibung *log* liefert den natürlichen Logarithmus des Arguments *x* zurück.

logl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Der komplexe natürliche Logarithmus wird definiert durch die Gleichung:

$$log(z) = log(abs(z)) + i\ arg(z))$$

Rückgabewert Bei fehlerfreier Ausführung liefern *log* und *logl* den berechneten Wert $ln(x)$ zurück. Ist x ein reelles Argument und $x < 0$, wird die globale Variable *errno* auf folgenden Wert gesetzt:

EDOM Bereichsfehler

Ist $x = 0$, dann liefern die Funktionen den negativen Wert HUGE_VAL (*log*) bzw. _LHUGE_VAL (*logl*) zurück und setzen *errno* auf ERANGE.

Die Behandlung von Fehlern kann für diese Funktionen über die Routine *matherr* bzw. *_matherrl* modifiziert werden.

Siehe auch *complex, exp, log10, sqrt*

Beispiel

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double result;
   double x = 8.6872;

   result = log(x);
   printf("The natural log of %lf is %lf\n",x,result);

   return 0;
}
```

log10, log10l

Funktion Berechnet den Logarithmus des Arguments zur Basis 10.

Syntax

reelle Version:

```
#include <math.h>
double log10(double x);
long double log10l(long double x);
```

komplexe Version:

```
#include <complex.h>
complex log10(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*log10l*
x	x	x	x		*reelles log10*
x		x		x	*komplexes log10*

Beschreibung

log10 berechnet den Logarithmus *n* des Arguments *x* zur Basis 10, erfüllt also die Gleichung $x = 10^n$.

log10l ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Der komplexe Logarithmus zur Basis 10 wird definiert durch die Gleichung

$$log10(z) = log(z) / log(10)$$

Rückgabewert

log10 und *log10l* liefern das Ergebnis der Berechnung von $log_{10}(x)$ zurück.

Ist x ein reelles Argument und $x < 0$, so setzen *log10* und *log10l* die globale Variable *errno* auf den Wert

EDOM Bereichsfehler

Ist $x = 0$, dann liefern die Funktionen den negativen Wert HUGE_VAL (*log10*) bzw. _LHUGE_VAL (*log10l*) zurück.

Die Behandlung von Fehlern kann für diese Funktionen über die Routine *matherr* bzw. *_matherrl* modifiziert werden.

Siehe auch

complex, *exp*, *log*

Beispiel

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double result;
   double x = 800.6872;

   result = log10(x);
   printf("The common log of %lf is %lf\n",x,result);

   return 0;
}
```

longjmp

Funktion

Führt ein »nicht-lokales« *goto* aus.

Syntax

```
#include <setjmp.h>
void longjmp(jmp_buf jmpb, int retval);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung

setjmp speichert den aktuellen Task-Zustand in dem durch *jmpb* angegebenen Puffer. Ein Aufruf von *longjmp* zu einem späteren Zeitpunkt stellt diesen Zustand wieder her – die Task setzt die Arbeit an der Stelle fort, an der sich der Aufruf von *setjmp* befindet.

Der Aufruf von *setjmp* liefert immer den Wert 0 zurück, ein Sprung mit *longjmp* an die Aufrufstelle von *setjmp* mit *retval* (immer ungleich Null) wirkt so, als ob *setjmp* den Wert *retval* geliefert hätte.

Ein Task-Zustand besteht aus folgenden Komponenten:

- Alle Segmentregister (CS, DS, ES, SS);
- Register-Variablen (SI, DI);
- Stack-Zeiger (SP);
- Frame-Zeiger für lokale Variablen (BP);
- Flags.

Ein Task-Zustand ist vollständig genug, um mit *setjmp* und *longjmp* Coroutinen zu implementieren.

setjmp muß vor *longjmp* aufgerufen werden. Die Routine, die *setjmp* aufgerufen und *jmpb* gesetzt hat, muß immer noch aktiv sein, bevor *longjmp* aufgerufen wird. Andernfalls sind die Ergebnisse unvorhersehbar.

longjmp kann in *retval* nicht den Wert 0 übergeben. Gegebenenfalls wird *retval* automatisch auf 1 geändert.

Wenn Sie mit Overlays arbeiten, dürfen Sie *setjmp* und *longjmp* nicht verwenden, um Coroutinen zu implementieren. Zwar sichert *setjmp* alle von Coroutinen benötigten Register und *longjmp* stellt sie wieder her, aber die Overlay-Verwaltung muß den Inhalt des Stack überwachen, und sie geht davon aus, daß es nur einen Stack gibt. Implementieren Sie aber Coroutinen, so gibt es zwei Stacks oder zwei Teile desselben Stack, und die Overlay-Verwaltung wird sie nicht korrekt behandeln.

Sie können durchaus mit Hintergrundroutinen arbeiten, diese dürfen aber keinerlei Overlay-Code benutzen. Sie dürfen auch nicht mit *setjmp* und *longjmp* von oder zu Hintergrundroutinen wechseln, wenn Sie mit Overlays arbeiten. Wenn Sie in Ihren Hintergrundroutinen keinen Overlay-Code verwenden, wird die Overlay-Verwaltung durch die Existenz eines Hintergrund-Stack aber nicht gestört.

Rückgabewert

Keiner.

Siehe auch

ctrlbrk, *setjmp*, *signal*

Beispiel

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>
void subroutine(jmp_buf);
int main(void)
{

   int value;
   jmp_buf jumper;
   value = setjmp(jumper);
   if (value != 0)
   {
      printf("Longjmp with value %d\n", value);
      exit(value);
   }
   printf("About to call subroutine ... \n");
   subroutine(jumper);
   return 0;
}
void subroutine(jmp_buf jumper)
{
   longjmp(jumper,1);
}
```

Programm-ausgabe:

```
About to call subroutine ...
Longjmp with value 1
```

lowvideo

Funktion — Setzt das Attribut »niedrige Intensität« für Zeichenausgaben.

Syntax

```
#include <conio.h>
void lowvideo(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *lowvideo* löscht das Bit für »hohe Intensität« der aktuellen Vordergrundfarbe. Bereits ausgegebene Zeichen bleiben unverändert – die Funktion legt lediglich das Zeichenattribut für folgende Ausgaben über die Routinen von conio.h fest (d.h. für *cprintf, cputs* usw).

Rückgabewert — Keiner.

Siehe auch — *highvideo, normvideo, textattr, textcolor*

Beispiel

```
#include <conio.h>

int main(void)
```

```
{
   clrscr();

   highvideo();
   cprintf("High Intensity Text\r\n");
   lowvideo();
   gotoxy(1,2);
   cprintf("Low Intensity Text\r\n");

   return 0;
}
```

_lrotl

Funktion — Rotiert einen vorzeichenlosen **long**-Wert bitweise nach links.

Syntax

```
#include <stdlib.h>
unsigned long _lrotl(unsigned long val, int count);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *_lrotl* rotiert den als *val* übergebenen Wert um *count* Bitpositionen nach links. *val* ist ein vorzeichenloser Integer vom Typ **long**.

Rückgabewert — *_lrotl* liefert den um *count* Bitpositionen linksrotierten Wert von *val* zurück.

Siehe auch — *_lrotr, _rotl, _rotr*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

/* function prototypes */

int lrotl_example(void);
int lrotr_example(void);

/* lrotl example */

int lrotl_example(void)
{
   unsigned long result;
   unsigned long value = 100;

   result = _lrotl(value,1);
   printf("The value %lu rotated left one bit is: "
          "%lu\n", value, result);

   return 0;
}
```

```
/* lrotr example */
int lrotr_example(void)
{
   unsigned long result;
   unsigned long value = 100;

   result = _lrotr(value,1);
   printf("The value %lu rotated right one bit is: "
          "%lu\n", value, result);

   return 0;
}

int main(void)
{
   lrotl_example();
   lrotr_example();
   return 0;
}
```

_lrotr

Funktion

Rotiert einen vorzeichenlosen **long**-Wert bitweise nach rechts.

Syntax

```
#include <stdlib.h>
unsigned long _lrotr(unsigned long val, int count);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

_lrotr rotiert den als *val* übergebenen Wert um *count* Bitpositionen nach rechts. *val* ist ein vorzeichenloser Integer vom Typ **long**.

Rückgabewert

_lrotr liefert den um *count* Bitpositionen rechtsrotierten Wert von *val* zurück.

Siehe auch

_lrotl, _rotl, _rotr

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

/* function prototypes */

int lrotl_example(void);
int lrotr_example(void);

/* lrotl example */

int lrotl_example(void)
{
   unsigned long result;
   unsigned long value = 100;
```

```
   result = _lrotl(value,1);
   printf("The value %lu rotated left one bit is: "
          "%lu\n", value, result);

   return 0;
}

/* lrotr example */

int lrotr_example(void)
{
   unsigned long result;
   unsigned long value = 100;

   result = _lrotr(value,1);
   printf("The value %lu rotated right one bit is: "
          "%lu\n", value, result);

   return 0;
}

int main(void)
{
   lrotl_example();
   lrotr_example();
   return 0;
}
```

lsearch

Funktion Führt eine lineare Suche durch.

Syntax

```
#include <stdlib.h>
void *lsearch(const void *key, void *base,
              size_t *num, size_t width,
              int (*fcmp)(const void *, const void *));
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *lsearch* dient zum Durchsuchen von Tabellen. Da die Suche hier *linear* stattfindet, können die Elemente im Gegensatz zu *bsearch* auch in unsortierter Form vorliegen.

Im Unterschied zu *bsearch* und *lfind*, die lediglich eine Suche ausführen, fügt *lsearch* das angegebene Element an die Liste an, falls es nicht bereits in ihr enthalten ist.

Der Parameter *base* muß auf das erste Element der Tabelle zeigen, *num* auf einen Integer, der die Anzahl der Elemente enthält. *width* gibt die Größe eines einzelnen Elements in Bytes an, *key* zeigt auf die zu suchende Information.

Der Parameter *fcmp* zeigt auf eine vom Programmierer

erstellte Routine, der jeweils zwei Argumente in Form von Zeigern (**key*, **elem*) zum Vergleich übergeben werden: *key* (die zu suchende Information) und *elem* (das Element, mit dem der Vergleich stattfinden soll).

Auf welche Art und Weise die Vergleichsroutine arbeitet, ist allein Sache des Programmierers.

Rückgabewert Die Funktion liefert die Adresse des ersten Elements der Tabelle zurück, das mit den gesuchten Daten identisch ist. Wenn **key* ungleich **elem* ist, muß *fcmp* einen Wert ungleich Null zurückliefern, ansonsten Null.

Siehe auch *bsearch*, *lfind*, *qsort*

Beispiel

```
/* lsearch example */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>     /* for strcmp declaration */

/* initialize number of colors */
char *colors[10] = { "Red", "Blue", "Green" };
int ncolors = 3;

int colorscmp(char **arg1, char **arg2)
{
   return(strcmp(*arg1, *arg2));
}

int addelem(char *key)
{
   int oldn = ncolors;
   lsearch(key, colors, (size_t *)&ncolors,
           sizeof(char *),
     (int(*)(const void *,const void *))colorscmp);
   return(ncolors == oldn);
}

int main(void)
{
   int i;
   char *key = "Purple";

   if (addelem(key))
      printf("%s already in colors table\n", key);
   else
   {
      strcpy(colors[ncolors-1],key);
      printf("%s added to colors table\n", key);
   }
```

```
   printf("The colors:\n");
   for (i = 0; i < ncolors; i++)
      printf("%s\n", colors[i]);
   return 0;
}
```

lseek

Funktion

Setzt die aktuelle Position des Dateizeigers innerhalb einer Datei.

Syntax

```
#include <io.h>
long lseek(int handle, long offset, int fromwhere);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

lseek setzt die Position des Dateizeigers innerhalb der durch *handle* bezeichneten Datei.

Der Parameter *offset* gibt jeweils die zu setzende Entfernung in Bytes an, die Positionierung wird über drei in io.h definierte Konstanten für *fromwhere* festgelegt:

SEEK_SET (Wert 0) – relativ zum Dateianfang
SEEK_CUR (Wert 1) – relativ zur aktuellen Position
SEEK_END (Wert 2) – relativ zum Dateiende

Rückgabewert

lseek liefert bei fehlerfreier Ausführung den Offset-Wert der neuen Zeigerposition, gemessen in Bytes vom Datenanfang, zurück. Im Falle eines Fehlers wird der Wert -1L zurückgeliefert, und die globale Variable *errno* bekommt einen der folgenden Werte:

EBADF Ungültiges Handle
EINVAL Ungültiges Argument

Bei der Anwendung auf Gerätedateien (wie Terminals, Druckern, Schnittstellen usw.) ist der Rückgabewert undefiniert.

Siehe auch

filelength, fseek, ftell, getc, open, sopen, ungetc, _write, write

Beispiel

```
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
```

```
   char msg[] = "This is a test";
   char ch;

   /* create a file */
   handle = open("TEST.$$$", O_CREAT | O_RDWR,
                 S_IREAD | S_IWRITE);

   /* write some data to the file */
   write(handle, msg, strlen(msg));

   /* seek to the begining of the file */
   lseek(handle, 0L, SEEK_SET);

   /* reads chars from the file until we hit EOF */
   do
   {
      read(handle, &ch, 1);
      printf("%c", ch);
   }  while (!eof(handle));

   close(handle);
   return 0;
}
```

ltoa

Funktion — Konvertiert einen **long**-Wert in einen String.

Syntax

```
#include <stdlib.h>
char *ltoa(long value, char *string, int radix);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *ltoa* konvertiert den übergebenen Wert *value* in einen nullterminierten String und speichert den Rückgabewert an der durch *string* bezeichneten Position. *ltoa* erwartet für *value* einen **long**-Wert.

Der Parameter *radix* legt die Basis fest, auf der die Konvertierung stattfinden soll – mögliche Werte liegen im Bereich von 2 bis 36.

Wenn *radix* den Wert 10 hat und *value* kleiner als Null ist, stellt *ltoa* dem Ergebnis ein Minuszeichen voran.

Hinweis: Der übergebene *string* muß in allen Fällen lang genug sein, um das Ergebnis der Konvertierung (inklusive des abschließenden Nullzeichens »\0«) aufnehmen zu können. *ltoa* liefert bis zu 33 Bytes zurück.

Rückgabewert — Die Funktion liefert einen Zeiger auf *string* zurück.

Siehe auch — *itoa, ultoa*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   char string[25];
   long value = 123456789L;

   ltoa(value,string,10);
   printf("number = %ld  string = %s\n",value,string);

   return 0;
}
```

_makepath

Funktion

Erzeugt einen Pfadnamen aus einzelnen Komponenten.

Syntax

```
#include <stdlib.h>
void _makepath(char *path, const char *drive,
               const char *dir, const char *name,
               const char *ext);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_makepath erzeugt einen Pfadnamen aus Einzelkomponenten.

Der neue Pfadname heißt

```
X:\DIR\SUBDIR\NAME.EXT
```

wobei gilt:

drive = X:
dir = \DIR\SUBDIR\
name = NAME
ext = .EXT

Ist *drive* leer oder NULL, wird keine Laufwerksbezeichnung in den Pfadnamen eingefügt. Fehlt nach *drive* der abschließende Doppelpunkt (:), wird er automatisch in den Pfadnamen eingefügt.

Ist *dir* leer oder NULL, wird kein Verzeichnis in den Pfadnamen eingefügt. Fehlt nach *drive* der abschließende Schrägstrich (\ oder /), wird automatisch ein umgekehrter Schrägstrich (Backslash) in den Pfadnamen eingefügt.

Ist *name* leer oder NULL, wird kein Dateiname in den Pfadnamen eingefügt.

Ist *ext* leer oder NULL, wird keine Namenserweiterung in den Pfadnamen eingefügt. Fehlt der Punkt (.), wird er automatisch in den Pfadnamen eingefügt.

_makepath setzt voraus, daß der mit *path* übergebene String eine ausreichende Größe für den konstruierten Pfadnamen hat. Die Maximalgröße ist _MAX_PATH, wobei _MAX_PATH in stdlib.h definiert ist.

_makepath und *_splitpath* verhalten sich invers. Die Anwendung von *_makepath* auf die durch *_splitpath* ermittelten Elemente eines vollständigen Pfadnamens ergeben wieder das Original.

Rückgabewert Keiner.

Siehe auch *_fullpath, _splitpath*

Beispiel

```
#include <dir.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char s[_MAX_PATH];
  char drive[_MAX_DRIVE];
  char dir[_MAX_DIR];
  char file[_MAX_FNAME];
  char ext[_MAX_EXT];

  getcwd(s,_MAX_PATH);
         /* get current working directory */
  if (s[strlen(s)-1] != '\\')
  strcat(s,"\\");
         /* append a trailing \ character */
  _splitpath(s,drive,dir,file,ext);
         /* split the string to separate elems */
  strcpy(file,"DATA");
  strcpy(ext,".TXT");
  _makepath(s,drive,dir,file,ext);
         /* merge everything into one string */
  puts(s);
         /* display resulting string */
  return 0;
}
```

malloc

Funktion Reserviert Hauptspeicher.

Syntax

```
#include <stdlib.h> oder #include <alloc.h>
void *malloc(size_t size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibun *malloc* belegt einen Speicherbereich von *size* Bytes auf dem Heap. Der Heap wird für die dynamische Belegung von Speicherbereichen variabler Größe benutzt. Viele Datenstrukturen – z.B. Bäume und Listen – belegen Speicher auf dem Heap.

In den kleinen Datenmodellen wird mit **near**-Zeigern gearbeitet, der Heap liegt zwischen dem oberen Ende des Datensegments und dem unteren Ende des Stack. Seine Größe beträgt maximal 64 KByte, wobei Borland C++ 256 Bytes direkt unterhalb der aktuellen Stack-Spitze für Funktionsaufrufe und für DOS reserviert.

In den großen Datenmodellen wird mit **far**-Zeigern gearbeitet, der Heap beginnt oberhalb des Stack und endet mit der höchsten vom System zur Verfügung gestellten Speicheradresse.

Rückgabewert *malloc* liefert bei fehlerfreier Ausführung einen Zeiger, der auf den neu belegten Speicherbereich zeigt. Wenn kein Speicherbereich ausreichender Größe zur Verfügung steht, dann ist der Rückgabewert NULL, es findet keine Initialisierung bzw. Veränderung statt.

Wenn der Parameter *size* für *malloc* den Wert 0 hat, ist der Rückgabewert NULL.

Siehe auch *allocmem*, *calloc*, *coreleft*, *farcalloc*, *farmalloc*, *free*, *realloc*

Beispiel

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>

int main(void)
{
   char *str;

   /* allocate memory for string */
   if ((str = (char *) malloc(10)) == NULL)
   {
      printf("Not enough memory to allocate "
             "buffer\n");
      exit(1);
         /* terminate program if out of memory */
   }

   /* copy "Hello" into string */
   strcpy(str, "Hello");

   /* display string */
   printf("String is %   s\n", str);
```

```
    /* free memory */
    free(str);

    return 0;
}
```

matherr, _matherrl

Funktion

Vom Programmierer modifizierbare Routinen zur Behandlung von Berechnungsfehlern.

Syntax

```
#include <math.h>
int matherr(struct exception *e);
int _matherrl(struct _exceptionl *e);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

matherr wird aufgerufen, wenn die *math*-Bibliothek einen Fehler erzeugt.

_matherrl ist die **long double** Version. Sie wird aufgerufen, wenn eine *math*-Funktion vom Typ **long double** einen Fehler erzeugt. Sie können *matherr* und *_matherrl* modifizieren und eigene Routinen zur Fehlerbehandlung schreiben (am Ende dieses Abschnitts finden Sie eine entsprechende Implementierung von *matherr*).

matherr und *_matherrl* sind besonders geeignet zum Aufspüren von Definitions- und Wertebereichsfehlern. Besondere Fehler der Gleitkommaberechnung (wie Division durch Null) werden nicht gefunden. Solche Fehler müssen mit *signal* abgefangen werden.

matherr und *_matherrl* können vom Programmierer für die Behandlung spezieller Fehler beliebig erweitert und modifiziert werden. Anstelle der Version in der C-Bibliothek wird dann die modifizierte Funktion verwendet.

Wenn die Routine den Fehler korrigieren kann, sollte sie *retval* entsprechend setzen und einen Wert ungleich Null als Rückgabewert zurückliefern. Falls der Fehler nicht korrigiert werden kann, muß ihr Rückgabewert der Wert 0 sein – die globale Variable *errno* wird dann gesetzt und eine Fehlermeldung ausgegeben.

Die Strukturtypen *exception* und *_exceptionl* sind in math.h folgendermaßen definiert:

```
struct exception {
   int    type;        /* Fehlerart */
   char   _FAR *name;  /* Funktionsname */
   double arg1,        /* erstes Argument */
```

```
        arg2,       /* zweites Argument (wenn
                    /* vorhanden) */
        retval;     /* Ergebnis */
};
struct _exceptionl {
   int    type;       /* Fehlerart */
   char   _FAR *name; /* Funktionsname */
   long double arg1,  /* erstes Argument */
               arg2,  /* zweites Argument (wenn
                      /* vorhanden) */
               retval; /* Ergebnis */
};
```

Das Element *name* zeigt auf einen nullterminierten String mit dem Namen der mathematischen Funktion, durch die der Fehler ausgelöst wurde.

Die Elemente *arg1* und *arg2* enthalten die Werte der Argumente, die der mathematischen Funktion übergeben wurden. Wenn die Funktion nur einen Wert erwartet, dann ist dieser Wert in *arg1* gespeichert.

Das Element *retval* enthält den Standardwert, den *matherr* oder *_matherr* zurückliefern. Dieser Wert kann vom Programmierer geändert werden.

Das Element *type* beschreibt die Art des Fehlers und ist in math.h als Aufzählungstyp *_mexcep* (math exception) definiert. Die folgenden Werte sind möglich:

DOMAIN – das Argument liegt außerhalb des gültigen Definitionsbereichs für die Funktion (wie beispielsweise *log(-1)*

SING – der Rückgabewert wäre eine mathematische Singularität (wie beispielsweise *pow(0, -2)*)

OVERFLOW – das Argument würde einen Rückgabewert erzeugen, dessen Wert größer als DBL_MAX (bzw. LDBL_MAX) ist und damit außerhalb des darstellbaren Bereichs liegt (wie beispielsweise *exp(1000)*)

UNDERFLOW – das Argument würde einen Rückgabewert erzeugen, dessen Wert kleiner als DBL_MIN (bzw. LDBL_MIN) MINDOUBLE ist und damit außerhalb des darstellbaren Bereichs liegt (wie beispielsweise *exp(-1000)*)

TLOSS – das Argument würde eine »Hausnummer« erzeugen – ein Rückgabewert, bei dem sämtliche Dezimalstellen ungenau sind (wie beispielsweise *sin(10e70)*).

Die Makros DBL_MAX, DBL_MIN, LDBL_MAX und LDBL_MIN sind in float.h definiert. Den Quelltext der vordefinierten Versionen von *matherr* und *_matherrl* finden Sie auf den Borland C++ Disketten.

Die UNIX-Versionen von *matherr* und *_matherrl* (Ausgabe

einer Meldung und Programmende) sind nicht ANSI-kompatibel. Sie finden die UNIX-Versionen dieser Routinen in den Dateien MATHERR.C und MATHERRL.C auf den Borland C++ Disketten.

Rückgabewert

Die vordefinierten Versionen von *matherr* und *_matherrl* liefern für UNDERFLOW und TLOSS den Wert 1 zurück – in allen anderen Fällen wird mit dem Wert 0 angezeigt, daß die Fehlerbedingung nicht beseitigt werden konnte.

Eine selbstgeschriebene Version von *matherr* oder *_matherrl* kann den Wert von *retval* beliebig verändern und sollte einen Rückgabewert ungleich Null zurückliefern, wenn der Fehler beseitigt worden ist. Die mathematische Funktion, durch deren Aufruf der Fehler verursacht wurde, nimmt den in *retval* gesetzten Wert als Ergebnis an.

Wenn *matherr* oder *_matherrl* den Wert 0 zurückliefern, erhält die globale Variable *errno* den Wert 0 und eine Fehlermeldung wird ausgegeben.

Liefern *matherr* oder *_matherrl* einen Wert ungleich Null zurück, wird *errno* nicht gesetzt und keine Fehlermeldung ausgegeben.

Beispiel

```
#include <math.h>
#include <string.h>
#include <stdio.h>

int matherr (struct exception *a)
{
  if (a->type == DOMAIN)
    if (!strcmp(a->name,"sqrt")) {
      a->retval = sqrt (-(a->arg1));
    return 1;
    }
  return 0;
}

int main(void)
{
  double x = -2.0, y;
  y = sqrt(x);
  printf("Matherr corrected value: %lf\n",y);
  return 0;
}
```

max

Funktion

Liefert den größeren von zwei Werten zurück.

Syntax

```
#include <stdlib.h>
(type) max(a,b);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Dieses Makro vergleicht zwei Werte *a* und *b* gleichen Typs und liefert den größeren der beiden Werte zurück.

Rückgabewert *max* liefert den größeren der beiden Werte zurück. Der Typ des Rückgabewertes hängt vom Typ der beiden als *a* und *b* übergebenen Werte ab.

Siehe auch *min*

Beispiel

```
/* max example */

#include <stdlib.h>
#include <stdio.h>

#ifdef __cplusplus

  int max (int value1, int value2);

  int max(int value1, int value2)
  {
     return ( (value1 > value2) ? value1 : value2);
  }

#endif

int main(void)
{
   int x = 5;
   int y = 6;
   int z;
   z = max(x, y);
   printf("The larger number is %d\n", z);
   return 0;
}
```

Programmausgabe:

```
The larger number is 6
```

mblen

Funktion Ermittelt die Länge eines aus mehreren Bytes bestehenden Zeichens.

Syntax

```
#include <stdlib.h>
int mblen(const char *s, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung Ist *s* ungleich NULL, prüft *mblen* das Zeichen, auf das *s* zeigt. Der Parameter *n* legt fest, wieviele Bytes maximal geprüft werden.

Das Verhalten von *mblen* ist von der Einstellung der LC_CTYPE-Kategorie in der aktuellen locale-Variablen abhängig.

Rückgabewert *mblen* liefert einen Wert ungleich NULL zurück, wenn *s* den Wert NULL hat und das zu untersuchende Zeichen statusabhängige Codierungen enthält. Andernfalls liefert *mblen* den Wert NULL.

Ist *s* ungleich NULL, kann *mblen* folgende Rückgabewerte liefern:

- 0, wenn *s* auf das Null-Zeichen zeigt;
- -1, wenn die nächsten *n* Bytes kein gültiges Zeichen aus mehreren Bytes darstellen;
- die Anzahl der Bytes, aus der ein gültiges Zeichen besteht.

Siehe auch *mbtowc*, *mbstowc*, *setlocale*

mbstowcs

Funktion Konvertiert einen Multibyte-String in ein *wchar_t*-Array.

Syntax
```
#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *mbstowcs* konvertiert den Multibyte-String *s* in das durch *pwcs* bezeichnete Array. In diesem Array werden maximal *n* Werte gespeichert. Trifft *mbstowcs* auf eine ungültige Multibyte-Sequenz, wird *(size_t)-1* zurückgeliefert.

Wenn *mbstowcs* den Rückgabewert *n* hat, wird das Array *pwcs* nicht durch ein Nullzeichen abgeschlossen.

Rückgabewert *mbstowcs* liefert *(size_t)-1* zurück, wenn sie auf eine ungültige Multibyte-Sequenz trifft. Andernfalls wird die Anzahl der geänderten Array-Elemente zurückgeliefert (das abschließende Zeichen ist – falls vorhanden – darin nicht enthalten).

Siehe auch *mblen*, *mbtowc*, *setlocale*

mbtowc

Funktion Konvertiert ein Multibyte-Zeichen in einen Wert des Typs *wchar_t*.

Syntax

```
#include <stdlib.h>
int mbtowc (wchar_t *pwc, const char *s, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung Ist *s* ungleich NULL, stellt *mbtowc* die Anzahl der Bytes fest, aus der das durch *s* angegebene Multibyte-Zeichen besteht. Anschließend prüft *mbtowc* den Wert des Typs *wchar_t*, das dem Multibyte-Zeichen entspricht. Besteht eine Übereinstimmung zwischen *wchar_t* und dem Multibyte-Zeichen und ist *pwc* ungleich NULL, wird der Wert von *wchar_t* in dem Array gespeichert, auf das *pwc* zeigt. Der Parameter *n* legt fest, auf wieviele Bytes maximal geprüft wird.

Rückgabewert Zeigt *s* auf ein ungültiges Multibyte-Zeichen, wird -1 zurückgeliefert. Zeigt *s* auf das Zeichen NULL, hat *mbtowc* den Rückgabewert Null. In allen anderen Fällen liefert *mbtowc* die Anzahl der Bytes zurück, aus denen sich das konvertierte Multibyte-Zeichen zusammensetzt.

Der Rückgabewert ist nie größer als MB_CUR_MAX oder der Wert von *n*.

Siehe auch *mblen, mbstowcs, setlocale*

memccpy, _fmemccpy

Funktion Kopiert einen Block mit *n* Bytes.

Syntax

```
#include <mem.h>
```

Near-Version:

```
void *memccpy(void *dest, const void *src,
              int c, size_t n)
```

Far-Version:

```
void far * far _fmemccpy(void far *dest,
                         const void far *src,
                         int c, size_t n)
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x			*Near-Version*
x		x			*Far-Version*

Beschreibung *memccpy* ist unter UNIX System V verfügbar. Die Funktion kopiert einen Block mit *n* Bytes von *src* nach *dest*. Der Kopiervorgang wird entweder nach *n* Bytes oder *nach* der Kopie des ersten Bytes mit dem Wert *c* beendet – je nachdem, welche Bedingung zuerst zutrifft.

Rückgabewert *memccpy* liefert NULL zurück, wenn *n* Bytes kopiert wurden (und kein Byte des Wertes *c* dabei gefunden wurde). Wurde *memccpy* dagegen durch die Kopie eines Bytes mit dem Wert *c* beendet, dann wird ein Zeiger zurückgeliefert, der auf das nächste Byte nach dem kopierten Wert *c* in *dest* zeigt.

Siehe auch *memcpy, memmove, memset*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char *src = "This is the source string";
   char dest[50];
   char *ptr;

   ptr = (char *) memccpy(dest, src, 'c',strlen(src));

   if (ptr)
   {
      *ptr = '\0';
      printf("The character was found: %s\n", dest);
   }
   else
      printf("The character wasn't found\n");
   return 0;
}
```

memchr, _fmemchr

Funktion Sucht *n* Bytes nach dem Wert *c* ab.

Syntax

```
#include <mem.h>
Near-Version:
void *memchr(const void *s, int c, size_t n);

Far-Version:
void far * far _fmemchr(const void far *s, int c,
                        size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *memchr* ist unter UNIX Sytem V verfügbar. Die Funktion sucht die ersten *n* Bytes des Blocks *s* nach dem Wert *c* ab.

Rückgabewert *memchr* liefert die Adresse der ersten Fundstelle von *c* im Array *s* zurück – oder NULL, wenn *c* innerhalb der ersten *n* Bytes von *s* nicht vorkommt.

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char str[17];
   char *ptr;

   strcpy(str, "This is a string");
   ptr = (char *) memchr(str, 'r', strlen(str));
   if (ptr)
      printf("The character 'r' is at position: %d\n",
             ptr - str);
   else
      printf("The character was not found\n");
   return 0;
}
```

memcmp, _fmemcmp

Funktion Vergleicht die ersten *n* Bytes zweier Blöcke.

Syntax

```
#include <mem.h>
```

Near-Version:

```
int memcmp(const void *s1, const void *s2, size_t n);
```

Far-Version:

```
int far * far _fmemcmp(const void far*s1,
                       const void far *s2, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *memcmp* ist unter UNIX System V verfügbar. Die Funktion vergleicht die ersten *n* Bytes der Blöcke *s1* und *s2* miteinander. *memcmp* behandelt die Bytes als **unsigned char** .

Rückgabewert *memcmp* bricht den Vergleich beim ersten nicht übereinstimmenden Zeichen ab. Das Vergleichsergebnis wird als **int**-Wert zurückgeliefert:

<0	wenn *s1* kleiner als *s2* ist
$==0$	wenn die ersten *n* Zeichen von *s1* und *s2* gleich sind
>0	wenn *s1* größer als *s2* ist

Beispielsweise liefert `memcmp("\xFF", "x7F", 1)` einen Wert größer Null.

Siehe auch *memicmp*

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *buf1 = "aaa";
   char *buf2 = "bbb";
   char *buf3 = "ccc";

   int stat;

   stat = memcmp(buf2, buf1, strlen(buf2));
   if (stat > 0)
      printf("buffer 2 is greater than buffer 1\n");
   else
      printf("buffer 2 is less than buffer 1\n");

   stat = memcmp(buf2, buf3, strlen(buf2));
   if (stat > 0)
      printf("buffer 2 is greater than buffer 3\n");
   else
      printf("buffer 2 is less than buffer 3\n");

   return 0;
}
```

memcpy, _fmemcpy

Funktion Kopiert *n* Bytes von *src* nach *dest*.

Syntax

```
#include <mem.h>
```

Near-Version:

```
void *memcpy(void *dest, const void *src, size_t n);
```

Far-Version:

```
void far * far _fmemcpy(void far *dest,
                        const void far *src, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *memcpy* ist unter UNIX System V verfügbar. Die Funktion kopiert *n* Bytes von *src* nach *dest*. Überlappen sich *scr* und *dest*, ist das Verhalten von *memcpy* undefiniert.

Rückgabewert *memcpy* liefert den übergebenen Parameter *dest* zurück.

Siehe auch *memccpy, memmove, memset, movedata, movmem*

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char src[] = "******************************";
   char dest[] = "abcdefghijlkmnopqrstuvwxyz"
                 "0123456709";
   char *ptr;

   printf("destination before memcpy: %s\n", dest);
   ptr = (char *) memcpy(dest, src, strlen(src));
   if (ptr)
      printf("destination after memcpy: %s\n", dest);
   else
      printf("memcpy failed\n");
   return 0;
}
```

memicmp, _fmemicmp

Funktion

Vergleicht die ersten *n* Bytes zweier Arrays, wobei zwischen Groß- und Kleinschreibung nicht unterschieden wird.

Syntax

```
#include <mem.h>
```

Near-Version:

```
int memicmp(const void *s1, const void *s2, size_t n);
```

Far-Version:

```
int far _fmemicmp(const void far *s1,
                  const void far *s2, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x			*Near-Version*
x		x			*Far-Version*

Beschreibung

memicmp ist umter UNIX System V verfügbar. Die Funktion vergleicht die ersten *n* Bytes der Blöcke *s1* und *s2* miteinander, wobei zwischen Groß- und Kleinschreibung nicht unterschieden wird.

Rückgabewert

memicmp bricht den Vergleich beim ersten nicht übereinstimmenden Zeichen ab. Das Vergleichsergebnis wird als **int** zurückgeliefert:

<0	wenn *s1* kleiner als *s2* ist
$==0$	wenn die ersten *n* Zeichen von *s1* und *s2* gleich sind
>0	wenn *s1* größer als *s2* ist

Siehe auch *memcmp*

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *buf1 = "ABCDE123";
   char *buf2 = "abcde456";
   int stat;
   stat = memicmp(buf1, buf2, 5);
   printf("The strings to position 5 are ");
   if (stat)
      printf("not ");
   printf("the same\n");
   return 0;
}
```

memmove

Funktion Kopiert einen Block von *n* Bytes.

Syntax

```
#include <mem.h>
void *memmove(void *dest, const void *src, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *memmove* ist unter UNIX System V verfügbar. Die Funktion kopiert *n* Bytes von *src* nach *dest* und ist daher im Prinzip identisch mit *memcpy* – hier werden jedoch Überlappungen von Quell- und Zielgebiet automatisch berücksichtigt.

Rückgabewert *memmove* liefert den übergebenen Parameter *dest* zurück, d.h. einen Zeiger auf den Zielblock.

Siehe auch *memccpy, memcpy, movmem*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
  char *dest = "abcdefghijklmnopqrstuvwxyz0123456789";
  char *src = "******************************";
  printf("destination prior to memmove: %s\n", dest);
  memmove(dest, src, 26);
  printf("destination after memmove:    %s\n", dest);
  return 0;
}
```

memset, _fmemset

Funktion

Setzt die ersten *n* Bytes des Speicherblocks *s* auf den Wert von *c*.

Syntax

```
#include <mem.h>
```

Near-Version:

```
void *memset(void *s, int c, size_t n);
```

Far-Version:

```
void far * far _fmemset(void far *s, int c, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung

memset ist unter UNIX System V verfügbar.

memset setzt die ersten *n* Bytes des Array *s* auf den Wert von *c*.

Rückgabewert

memset liefert den übergebenen Parameter *s* zurück.

Siehe auch

memccpy, memcpy, setmem

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <mem.h>

int main(void)
{
   char buffer[] = "Hello world\n";

   printf("Buffer before memset: %s\n", buffer);
   memset(buffer, '*', strlen(buffer) - 1);
   printf("Buffer after memset:  %s\n", buffer);
   return 0;
}
```

min

Funktion

Liefert den kleineren von zwei Werten zurück.

Syntax

```
#include <stdlib.h>
(type) min(a,b);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

min vergleicht zwei Werte *a* und *b* gleichen Typs und liefert diesen Typ auch als Rückgabewert.

Rückgabewert *min* gibt den kleineren der beiden Werte zurück. Der Typ des Rückgabewertes hängt vom Typ der beiden als *a* und *b* übergebenen Werte ab.

Siehe auch *max*

Beispiel

```
/* min example */

#include <stdlib.h>
#include <stdio.h>

#ifdef __cplusplus

  int min (int value1, int value2);

  int min(int value1, int value2)
  {
     return ( (value1 < value2) ? value1 : value2);
  }

#endif

int main()
{
   int x = 5;
   int y = 6;
   printf("The smaller number is %d\n", min(x,y));

   return 0;
}
```

Programmausgabe:

```
The smaller number is 5
```

mkdir

Funktion Erzeugt ein Verzeichnis.

Syntax

```
#include <dir.h>
int mkdir(const char *path);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *mkdir* ist unter UNIX System V verfügbar, benötigt aber einen zusätzlichen *filemode*-Parameter. *mkdir* erzeugt ein Verzeichnis mit dem in *path* angegebenen Namen.

Rückgabewert *mkdir* liefert den Wert 0 zurück, wenn die Operation fehlerfrei ausgeführt werden konnte.

Wenn ein Fehler auftritt, so ist der Rückgabewert -1 und die globale Variable *errno* bekommt einen der beiden folgenden Werte:

EACCES	Zugriff nicht gestattet
ENOENT	Datei oder Verzeichnis nicht gefunden

Siehe auch *chdir*, *getcurdir*, *getcwd*, *rmdir*

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>

int main(void)
{
  int status;

   clrscr();
   status = mkdir("asdfjklm");
   (!status) ? (printf("Directory created\n")) :
               (printf("Unable to create "
                       "directory\n"));

   getch();
   system("dir");
   getch();

   status = rmdir("asdfjklm");
   if (status == 0)
     printf("Directory deleted\n");
   else
     perror("Unable to delete directory");

   return 0;
}
```

MK_FP

Funktion Erzeugt einen **far**-Zeiger.

Syntax

```
#include <dos.h>
void far *MK_FP(unsigned seg, unsigned ofs);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *MK_FP* ist ein Makro, über das ein **far**-Zeiger aus seinen Komponenten *seg* und *ofs* erzeugt werden kann.

Rückgabewert *MK_FP* liefert einen **far**-Zeiger.

Siehe auch *FP_OFF*, *FP_SEG*, *movedata*, *segread*

Beispiel

```
#include <dos.h>
#include <graphics.h>

int main(void)
{
  int gd, gm, i;
  unsigned int far *screen;
  detectgrah(&gd, &qm);
  if (gd == HERCMONO)
      screen = MK_FP(0xB000, 0);
  else
      screen = MK_FP(0xB800, 0);
  for (i = 0; i < 26; i++)
      screen[i] = 0x0700 + ('a' + i);
  return 0;
}
```

mktemp

Funktion Erzeugt einen eindeutigen Dateinamen.

Syntax

```
#include <dir.h>
char *mktemp(char *template);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *mktemp* ersetzt den durch *template* bezeichneten String durch einen eindeutigen Dateinamen.

template sollte ein nullterminierter String mit sechs abschließenden »X«-Zeichen sein. Diese »X« werden, beginnend mit *AA.AAA*, durch zwei Buchstaben, einen Punkt und drei Buchstaben für die Dateinamenserweiterung ersetzt.

mktemp verwandelt *XXXXXX* als erstes in *AA.AAA* (macht also aus *tdatXXXXXX* den Namen *tdatAA.AAA*) und prüft dann innerhalb des angegebenen bzw. aktuellen Verzeichnisses, ob eine Datei dieses Namens bereits existiert. Wenn ja, wird der Versuch mit *AA.AAB*, *AA.AAC* usw. wiederholt – solange, bis sich der Name einer noch nicht existierenden Datei ergibt.

Rückgabewert Falls *template* mit sechs »X« endet und ein gültiges Verzeichnis angibt (bzw. nur einen Dateinamen enthält), verändert *mktemp template* entsprechend und liefert einen Zeiger auf diesen String zurück; ansonsten ist der Rückgabewert NULL. *template* bleibt unverändert.

Beispiel

```
#include <dir.h>
#include <stdio.h>

int main(void)
```

```
{
   /* fname defines the template for the
     temporary file.  */

   char *fname = "TXXXXXX", *ptr;

   ptr = mktemp(fname);
   printf("%s\n",ptr);
   return 0;
}
```

mktime

Funktion — Konvertiert die Uhrzeit in das Kalenderformat.

Syntax

```
#include <time.h>
time_t mktime(struct tm *t);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung — *mktime* konvertiert die Uhrzeit in der Struktur, auf die *t* zeigt, in eine Kalenderzeit mit dem Format der *time*-Funktion. Die ursprünglichen Werte der Felder *tm_sec*, *tm_min*, *tm_hour*, *tm_mday* und *tm_mon* sind nicht auf den durch die *tm*-Struktur festgelegten Bereich beschränkt. Befinden sich die Felder nicht im korrekten Bereich, werden sie angepaßt. Die Werte der Felder *tm_wday* und *tm_yday* werden nach der Anpassung der anderen Felder berechnet. Kann die Kalenderzeit nicht dargestellt werden, liefert *mktime* -1 zurück.

Die zulässige Bereich für die Kalenderzeit liegt zwischen dem 1. Januar 1970 00:00:00 und dem 19. Januar 2038 03:14:07.

Rückgabewert — Siehe Beschreibung.

Siehe auch — *localtime*, *strftime*, *time*

Beispiel

```
#include <stdio.h>
#include <time.h>

char *wday[] = {"Sunday", "Monday", "Tuesday",
                "Wednesday", "Thursday", "Friday",
                "Saturday", "Unknown"};

int main(void)
{
   struct tm time_check;
   int year, month, day;

/* Input  year, month and day to find weekday for */
   printf("Year:  ");
```

```
   scanf("%d", &year);
   printf("Month: ");
   scanf("%d", &month);
   printf("Day:   ");
   scanf("%d", &day);

/*  load the time_check structure with the data */
   time_check.tm_year = year - 1900;
   time_check.tm_mon  = month - 1;
   time_check.tm_mday = day;
   time_check.tm_hour = 0;
   time_check.tm_min  = 0;
   time_check.tm_sec  = 1;
   time_check.tm_isdst = -1;

/*  call mktime to fill in the weekday field
    of the structure
*/
   if (mktime(&time_check) == -1)
      time_check.tm_wday = 7;

/*  print out the day of the week */
   printf("That day is a %s\n",
          wday[time_check.tm_wday]);
   return 0;
}
```

modf, modfl

Funktion

Teilt einen **double**-oder **long double** -Wert in Ganzzahl und Bruchteil auf.

Syntax

```
#include <math.h>
double modf(double x, double *ipart);
long double modfl(long double x, long double *ipart);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*modf*
x		x			*modfl*

Beschreibung

modf liefert den Bruchteil des **double**-Wertes *x*, der ganzzahlige Anteil wird in **ipart* gespeichert.

modfl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert

modf und *modfl* liefern den Bruchteil des Arguments *x* zurück.

Siehe auch

fmod, ldexp

Beispiel

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double fraction, integer;
   double number = 100000.567;

   fraction = modf(number, &integer);
   printf("The whole and fractional parts of %lf "
          "are %lf and %lf\n", number, integer,
          fraction);
   return 0;
}
```

movedata

Funktion Kopiert Daten.

Syntax

```
#include <mem.h>
void movedata(unsigned srcseg, unsigned srcoff,
              unsigned dstseg, unsigned dstoff,
              size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *movedata* kopiert *n* Bytes von einer Quelladresse (*srcseg: srcoff*) zu einer Zieladresse (*dstseg:dstoff*). *movedata* funktioniert unabhängig vom verwendeten Speichermodell.

Rückgabewert Keiner.

Siehe auch *FP_OFF, memcpy, MK_FP, movmem, segread*

Beispiel

```
#include <mem.h>

#define MONO_BASE 0xB000

char buf[80*25*2];

/* saves contents of monochrome screen in buffer */
void save_mono_screen(char near *buffer)
{
   movedata(MONO_BASE, 0, _DS, (unsigned)buffer,
            80*25*2);
}

int main(void)
{
   save_mono_screen(buf);
}
```

moverel

Funktion Bewegt die aktuelle Position (CP) um eine relative Entfernung.

Syntax
```
#include <graphics.h>
void far moverel(int dx, int dy);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *moverel* bewegt die aktuelle Position (CP) um *dx* Pixel in der x-Richtung und um *dy* Pixel in der y-Richtung.

Rückgabewert Keiner.

Siehe auch *moveto*

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* move the C.P. to location (20, 30) */
   moveto(20, 30);

   /* plot a pixel at the C.P. */
   putpixel(getx(), gety(), getmaxcolor());

   /* create and output a message at (20, 30) */
   sprintf(msg, " (%  d, %d)", getx(), gety());
```

```
   outtextxy(20, 30, msg);

   /* move to a point a relative distance */
   /* away from the current value of C.P. */
   moverel(100, 100);

   /* plot a pixel at the C.P. */
   putpixel(getx(), gety(), getmaxcolor());

   /* create and output a message at C.P. */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

movetext

Funktion

Kopiert einen rechteckigen Ausschnitt des Textbildschirms in einen anderen Ausschnitt.

Syntax

```
#include <conio.h>
int movetext(int left, int top, int right, int bottom,
             int destleft, int desttop);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

movetext kopiert einen rechteckigen Ausschnitt des Textbildschirms, dessen obere linke Ecke durch (*left*, *top*) und dessen untere rechte Ecke durch (*right*, *bottom*) festgelegt ist, in einen rechteckigen Ausschnitt gleicher Größe, dessen obere linke Ecke mit (*destleft*, *desttop*) angegeben wird.

movetext arbeitet mit absoluten Bildschirm-Koordinaten, d.h. unabhängig von einem eventuell gesetzten Textfenster. Überlappende Ausschnitte werden korrekt kopiert.

movetext funktioniert nur im Textmodus und schreibt direkt in den Bildschirmspeicher.

Rückgabewert

Bei fehlerfreier Ausführung liefert *movetext* einen Wert ungleich Null zurück. Wenn *movetext* nicht ausgeführt werden konnte, weil entweder der »Quellausschnitt« oder der »Zielausschnitt« außerhalb des Bereichs des aktuellen Bildschirmmodus liegen, ist der Rückgabewert 0.

Siehe auch

gettext, *puttext*

Beispiel

```
#include <conio.h>
#include <string.h>

int main(void)
{
   char *str = "This is a test string";

   clrscr();
   cputs(str);
   getch();

   movetext(1, 1, strlen(str), 2, 10, 10);
   getch();

   return 0;
}
```

moveto

Funktion Setzt die aktuelle Position (CP) auf einen bestimmten Punkt.

Syntax

```
#include <graphics.h>
void far moveto(int x, int y);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *moveto* setzt die aktuelle Position (CP) auf den Punkt (*x*,*y*) im Zeichenfenster.

Rückgabewert Keiner.

Siehe auch *moverel*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
```

```
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
    }

    /* move the C.P. to location (20, 30) */
    moveto(20, 30);

    /* plot a pixel at the C.P. */
    putpixel(getx(), gety(), getmaxcolor());

    /* create and output a message at (20, 30) */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtextxy(20, 30, msg);

    /* move to (100, 100) */
    moveto(100, 100);

    /* plot a pixel at the C.P. */
    putpixel(getx(), gety(), getmaxcolor());

    /* create and output a message at C.P. */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtext(msg);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

movmem

Funktion: Verlagert Daten von einem Bereich in einen anderen.

Syntax:

```
#include <mem.h>
void movmem(void *src, void *dest, unsigned length);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung: *movmem* verlagert *length* Bytes von *src* nach *dest*. Quell- und Zielbereich können sich überlappen – *movmem* kümmert sich automatisch um die damit verbundenen Probleme.

Rückgabewert: Keiner.

Siehe auch: *memcpy, memmove, movedata*

Beispiel

```
#include <mem.h>
#include <alloc.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *source = "Borland International";
   char *destination;
   int length;

   length = strlen(source);
   destination = (char *) malloc(length + 1);
   movmem(source, destination, length);
   printf("%s\n", destination);

   return 0;
}
```

norm

Funktion Liefert das Quadrat des absoluten Werts.

Syntax

```
#include <complex.h>
double norm(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung *norm* liefert das Quadrat des absoluten Werts. Diese Funktion führt zu einem Überlauf, wenn der Realteil oder der Imaginärteil zu groß ist.

Rückgabewert *norm*(*x*) liefert *real*(*x*) * *real*(*x*) + *imag*(*x*) * *imag*(*x*) zurück.

Siehe auch *arg, complex, polar*

Beispiel

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary real part = "
        << imag(z) << "\n";
   cout << "z has complex conjugate = "
        << conj(z) << "\n";

   double mag = sqrt(norm(z));
```

```
   double ang = arg(z);
   cout << "The polar form of z is:\n";
   cout << "   magnitude = " << mag << "\n";
   cout << "   angle (in radians) = " << ang << "\n";
   cout << "Reconstructing z from its polar form "
           "gives:\n";
   cout << "   z = " << polar(mag,ang) << "\n";
   return 0;
}
```

normvideo

Funktion — Setzt das Textattribut auf normale Intensität.

Syntax

```
#include <conio.h>
void normvideo(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *normvideo* setzt das Attribut (Vorder- und Hintergrund), das beim Start des Programms gesetzt war.

Bereits *ausgegebene* Zeichen bleiben unverändert – die Funktion legt lediglich das Zeichenattribut für folgende Ausgaben über die Routinen von conio.h fest (d.h. für *cprintf*, *cputs* usw).

Rückgabewert — Keiner.

Siehe auch — *highvideo, lowvideo, textattr, textcolor*

Beispiel

```
#include <conio.h>

int main(void)
{
   normvideo();
   cprintf("NORMAL Intensity Text\r\n");
   return 0;
}
```

nosound

Funktion — Schaltet den eingebauten Lautsprecher ab.

Syntax

```
#include <dos.h>
void nosound(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Ein mit *sound* eingeschalteter Lautsprecher kann durch einen Aufruf von *nosound* wieder abgeschaltet werden.

Rückgabewert Keiner.

Siehe auch *delay, sound*

Beispiel

```
/* Emits a 7-Hz tone for 10 seconds.

     True story: 7 Hz is the resonant
     frequency of a chicken's skull cavity.
     This was determined empirically in
     Australia, where a new factory
     generating 7-Hz tones was located too
     close to a chicken ranch: When the
     factory started up, all the chickens
     died.

     Your PC may not be able to emit a 7-Hz tone. */

#include <dos.h>

int main(void)
{
   sound(7);
   delay(10000);
   nosound();
}
```

_open, _dos_open

Funktion Öffnet eine Datei für Lese- oder Schreibaktionen.

Syntax

```
#include <fcntl.h>
int _open(const char *filename, int oflags);

#include <fcntl.h>
#include <share.h>
#include <dos.h>
int _dos_open(const char *filename, unsigned oflags
              int *handlep);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_open* und *_dos_open* öffnen die durch *filename* angegebene Datei und bereiten die durch den Wert von *oflags* angegebenen Lese- und/oder Schreibaktionen vor. Die Datei wird immer binär geöffnet. Das Datei-Handle wird an der durch *handlep* angegebenen Position gespeichert.

oflags muß einen der folgenden Werte enthalten:

O_RDONLY	Nur Lesen
O_WRONLY	Nur Schreiben
O_RDWR	Lesen und Schreiben

Ab der DOS-Version 3.0 können auch folgende Werte über eine ODER-Operation in *oflags* verwendet werden:

O_NOINHERIT	Datei kann nicht an untergeordnete Prozesse übergeben werden.
SH_COMPAT	Ermöglicht weitere Zugriffe auf diese Datei mit SH_COMPAT. Wurde die Datei bereits in einem anderen SHARED-Modus geöffnet, ergibt der Aufruf eine Fehler.
SH_DENYRW	Ermöglicht den Zugriff auf die Datei nur über das aktuelle Handle.
SH_DENYWR	Über andere Handles kann diese Datei nur gelesen, aber nicht beschrieben werden.
SH_DENYRD	Über andere Handles kann diese Datei nur beschrieben werden.
SH_DENYNONE	Sämtliche Zugriffe auf diese Datei sind erlaubt, jedoch nicht solche mit SH_COMPAT.

Hinweis: Diese symbolischen Konstanten sind in fcntl.h und in share.h definiert.

In einem Aufruf von *_dos_open* oder *_open* kann jeweils nur eine der SH_DENYxxx-Konstanten angegeben werden – sie muß zusätzlich mit O_RDONLY, O_WRONLY oder O_RDWR über eine ODER-Operation verbunden sein.

Die maximal mögliche Anzahl offener Dateien wird durch HANDLE_MAX definiert.

Rückgabewert Wenn die Datei fehlerfrei geöffnet werden konnte, liefert *_open* das Handle dieser Datei zurück (einen **int**-Wert größer 0). *_dos_open* liefert bei fehlerfreier Ausführung 0 zurück und speichert das Datei-Handle in dem durch *handlep* angegebenen Bereich. Der Dateizeiger – er markiert die aktuelle Position in der Datei – wird auf den Dateianfang gesetzt.

Im Fehlerfall liefert *_open* den Wert -1 und *_dos_open* den DOS-Fehlercode zurück. Beide Funktionen setzen die globale Variable *errno* auf einen der folgenden Werte:

ENOENT	Suchweg nicht vorhanden (oder Dateiname ungültig)
EMFILE	Zu viele Dateien offen
EACCES	Zugriff nicht gestattet
EINVACC	Ungültiger Zugriffscode

Siehe auch *open*, *_read*, *sopen*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = _open("TEST.$$$", O_RDWR)) == -1)
   {
      perror("Error:");
      return 1;
   }
   _write(handle, msg, strlen(msg));
   _close(handle);
   return 0;
}
```

Beispiel 2

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <dos.h>

int main(void)
{
  int handle;
  unsigned nbytes;
  char msg[] = "Hello world\n";
  if (_dos_open("TEST.$$$", O_RDWR, &handle) != 0) {
    perror("Unable to open TEST.$$$");
    return 1;
  }
  if (_dos_write(handle,msg,strlen(msg),&nbytes) != 0)
    perror("Unable to write to TEST.$$$");
  printf("%u bytes written to TEST.$$$\n",nbytes);
  _dos_close(handle);
  return 0;
}
```

open

Funktion Öffnet eine Datei für Lese- oder Schreibaktionen.

Syntax

```
#include <fcntl.h>
#include <sys\stat.h>
int open(const char *path,
            int access [, unsigned mode]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

open öffnet die durch *path* angegebene Datei und bereitet – je nach den für *access* angegebenen Werten – folgende Lese- und/oder Schreibaktionen vor.

Um beim Erzeugen einer Datei einen bestimmten Zugriffsmodus festzulegen, kann die globale Variable *_fmode* verwendet oder ein Aufruf von *open* mit den Optionen O_CREAT und O_TRUNC und dem gewünschten Zugriffsmodus durchgeführt werden. Beispielsweise erzeugt der Aufruf

```
open("xmp", O_CREAT|O_TRUNC|O_BINARY,S_IREAD)
```

die Datei XMP im binären Modus für einen Nur-Lesen-Zugriff und kürzt Sie – falls sie bereits existiert – auf 0 Bytes.

Für den Parameter *access* sind zwei Arten von Konstanten in fcntl.h definiert, die über eine ODER-Operation miteinander verbunden werden müssen. Aus der ersten der beiden folgenden Listen muß genau ein Flag verwendet werden. Die restlichen Flags können in jeder logisch gültigen Kombination verwendet werden.

1. Flags für Lesen und/oder Schreiben

O_RDONLY	Nur Lesen
O_WRONLY	Nur Schreiben
O_RDWR	Lesen und Schreiben

2. Weitere Zugriff-Flags

O_NDELAY	Nur aus Gründen der UNIX-Kompatibilität definiert, unter DOS nicht verwendet.
O_APPEND	Neu geschriebene Daten werden grundsätzlich angehängt, das Überschreiben bereits existierender Daten ist nicht möglich.
O_CREAT	Wenn die Datei noch nicht existiert, wird sie mit den über mode angegebenen Attributen erzeugt. Falls die Datei bereits existiert, hat dieses Flag keine Wirkung.
O_TRUNC	Wenn die Datei bereits existiert, wird ihr vorheriger Inhalt gelöscht, die Dateiattribute bleiben unverändert.
O_EXCL	Nur zusammen mit O_CREAT. Falls die Datei bereits existiert, wird ein Fehler zurückgeliefert.
O_BINARY	Legt den Modus *binary* (unabhängig von *_fmode*) fest – gelesene bzw. geschriebene Daten werden nicht interpretiert.

O_TEXT	Legt den Modus *text* fest (unabhängig von *_fmode*).

Wenn Sie weder O_BINARY noch O_TEXT angeben, wird die Datei in dem in *_fmode* gesetzten Modus geöffnet.

Bei der Anwendung von O_CREAT (d.h. der Erzeugung einer neuen Datei) muß der Parameter *mode* über die folgenden symbolischen Konstanten festgelegt werden, die in sys/stat.h definiert sind:

S_IWRITE	Schreiben erlaubt
S_IREAD	Lesen erlaubt
S_IREAD | S_IWRITE	Lesen und Schreiben erlaubt

Rückgabewert Wenn die Datei fehlerfrei geöffnet werden konnte, liefert *open* das *Handle* dieser Datei zurück (einen **int** mit einem Wert größer 0, über den weitere Operationen mit dieser Datei ausgeführt werden). Der Dateizeiger – er markiert die aktuelle Position in der Datei – wird auf den Dateianfang gesetzt.

Im Fehlerfall liefert die Funktion den Wert -1 zurück und setzt die globale Variable *errno* auf einen der folgenden Werte:

ENOENT	Suchweg nicht vorhanden (oder Dateiname ungültig)
EMFILE	Zu viele Dateien offen
EACCES	Zugriff nicht gestattet
EINVACC	Ungültiger Zugriffscode

Siehe auch *chmod*, *chsize*, *close*, *_creat*, *creat*, *creatnew*, *createtemp*, *dup*, *dup2*, *fdopen*, *filelength*, *fopen*, *freopen*, *getftime*, *lock*, *lseek*, *_open*, *read*, *sopen*, *_write*, *write*

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = open("TEST.$$$", O_CREAT | O_TEXT))
       == -1)
   {
      perror("Error:");
      return 1;
   }
   write(handle, msg, strlen(msg));
   close(handle);
   return 0;
}
```

opendir

Funktion Öffnet einen Verzeichnis-Stream für Leseoperationen.

Syntax
```
#include <dirent.h>
DIR *opendir(char *dirname);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *opendir* ist auf unter UNIX auf POSIX-Systemen verfügbar. Die Funktion öffnet den Verzeichnis-Stream *dirname* für Leseoperationen. Der Stream wird so gesetzt, daß der erste Eintrag des Verzeichnisses gelesen werden kann.

Ein Verzeichnis-Stream wird durch die in dirent.h definierte DIR-Struktur repräsentiert. Der Programmierer hat keinen Zugriff auf die Felder dieser Struktur. Es können gleichzeitig mehrere Verzeichnis-Streams geöffnet und gelesen werden. Während des Lesens können Verzeichniseinträge erstellt oder gelöscht werden.

Mit der Funktion *readdir* können aufeinanderfolgende Einträge aus einem Verzeichnis-Stream gelesen werden. Das Entfernen eines nicht mehr benötigten Verzeichnis-Stream erfolgt mit *closedir*.

Rückgabewert Bei fehlerfreier Ausführung liefert *opendir* einen Zeiger auf den Verzeichnis-Stream zurück, der in Aufrufen von *readdir*, *rewinddir* und *closedir* verwendet werden kann. Kann das Verzeichnis nicht geöffnet werden, liefert *opendir* den Wert NULL und setzt die globale Variable *errno* auf einen der folgenden Werte:

ENOENT	Das Verzeichnis existiert nicht.
ENOMEM	Nicht genug Speicher für die Reservierung eines DIR-Objekts.

Siehe auch *closedir, readdir, rewinddir*

Beispiel
```
/* opendir.c - test opendir(),
   readdir(),closedir() */

#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>

void scandir(char *dirname)
{
  DIR *dir;
  struct dirent *ent;

  printf("First pass on '%   s':\n",dirname);
```

```
  if ((dir = opendir(dirname)) == NULL)
  {
    perror("Unable to open directory");
    exit(1);
  }
  while ((ent = readdir(dir)) != NULL)
    printf("%s\n",ent->d_name);

  printf("Second pass on '%s':\n",dirname);
  rewinddir(dir);
  while ((ent = readdir(dir)) != NULL)
    printf("%s\n",ent->d_name);
  if (closedir(dir) != 0)
    perror("Unable to close directory");
}

void main(int argc,char *argv[])
{
  if (argc != 2)
  {
    printf("usage: opendir dirname\n");
    exit(1);
  }
  scandir(argv[1]);
  exit(0);
}
```

outp

Funktion: Schreibt ein einzelnes Byte zu einer I/O-Adresse.

Syntax:

```
#include <conio.h>
int outp(unsigned portid, int value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung: Das Makro *outp* schreibt das niederwertige Byte von *value* zu der von *portid* angegebenen Adresse.

outp ist als inline-Makro definiert – der Compiler verwendet nur dann »echte« Funktionsaufrufe, wenn conio.h nicht aufgenommen wird (oder ein entsprechendes #*undef* eingefügt wird).

Rückgabewert: *outp* liefert *value* zurück.

Siehe auch: *inp, inpw, outpw*

Beispiel

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
  unsigned port = 0;
  int value;
  value = outp(port, 'C');
  printf("Value %c sent to port number %d\n",
         value, port);
  return 0;
}
```

outport, outportb

Funktion Schreibt ein WORD bzw. ein Byte zu einer I/O-Adresse.

Syntax

```
#include <dos.h>
void outport(int portid, int value);
void outportb(int portid, unsigned char value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *outport* arbeitet wie die 80x86-Anweisung *OUT*: das niederwertige Byte des durch *value* gegebenen WORD wird zu *portid* und das höherwertige Byte zu *portid +1* ausgegeben.

Das Makro *outportb* schreibt ein einzelnes Byte – gegeben durch *value* – zu der von *portid* angegebenen Adresse.

outportb ist als inline-Makro definiert – der Compiler verwendet nur dann »echte« Funktionsaufrufe, wenn dos.h nicht aufgenommen oder ein entsprechendes #*undef* eingefügt wird.

Rückgabewert Keiner.

Siehe auch *inport, inportb*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   int port = 0;
   int value = 'C';

   outport(port, value);
   printf("Value %d sent to port number %d\n",
          value, port);
   return 0;
}
```

outpw

Funktion Schreibt ein WORD zu einer I/O-Adresse.

Syntax
```
#include <conio.h>
unsigned outpw(unsigned portid, unsigned value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung Das Makro *outpw* schreibt das durch *value* gegebene 16-Bit-WORD zu der von *portid* bezeichneten Adresse. Über eine einzelne 16-Bit-*OUT*-Anweisung wird das niederwertige Byte von *value* zu *portid* und das höherwertige Byte zu *portid +1* ausgegeben.

outpw ist als inline-Makro definiert – der Compiler verwendet nur dann »echte« Funktionsaufrufe, wenn conio.h nicht aufgenommen oder ein entsprechendes #*undef* eingefügt wird.

Rückgabewert *outpw* liefert *value* zurück.

Siehe auch *inp, inpw, outp*

Beispiel
```
#include <stdio.h>
#include <conio.h>

int main(void)
{
  unsigned value;
  unsigned port = 0;
  value = outpw(port, 64);
  printf("Value %d sent to port number %d\n",
         value, port);
  return 0;
}
```

outtext

Funktion Gibt Text ab der aktuellen Position (CP) aus.

Syntax
```
#include <graphics.h>
void far outtext(char far *textstring);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *outtext* gibt den als *textstring* übergebenen String ab der aktuellen Position (CP) im Zeichenfenster aus. *outtext* benutzt die Schriftart und Formatierung, die durch *settextstyle* (Schriftart,

Rotation, Größe) und *settextjustify* (linksbündig, zentriert, rechtsbündig) vorgegeben ist.

Falls die horizontale Justierung LEFT_TEXT ist und die Textausgabe in horizontaler Richtung erfolgt (HORIZ_DIR), wird die X-Koordinate der aktuellen Position um *textwidth* (*textstring*) geändert, ansonsten bleibt sie unverändert.

Damit das Programm bei einem Wechsel der Schriftart und -größe nicht modifiziert werden muß, sollte der Platzbedarf auszugebender Strings immer über *textwidth* und *textheight* berechnet werden.

Ein automatischer Umbruch wie im Textmodus findet nicht statt: Ausgaben, die über den rechten Rand des Bildschirms bzw. des aktuellen Zeichenfensters hinausgehen würden, schneidet *outtext* einfach ab.

outtext funktioniert nur im Grafik-, nicht aber im Textmodus.

Rückgabewert Keiner.

Siehe auch *gettextsettings, outtextxy, settextjustify, textheight, textwidth*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* move the C.P. to the center of the screen */
   moveto(midx, midy);
```

```
    /* output text starting at the C.P. */
    outtext("This ");
    outtext("is ");
    outtext("a ");
    outtext("test.");

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

outtextxy

Funktion — Gibt Text ab einem bestimmten Punkt aus.

Syntax

```
#include <graphics.h>
void far outtextxy(int x, int y, char far *textstring);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *outtextxy* gibt den als *textstring* übergebenen String ab der Position (*x*,*y*) im Zeichenfenster aus. Die aktuelle Position wird dabei nicht verändert.

outtextxy benutzt die Schriftart und Formatierung, die durch *settextstyle* (Schriftart, Rotation, Größe) und *settextjustify* (linksbündig, zentriert, rechtsbündig) vorgegeben ist.

Damit das Programm bei einem Wechsel der Schriftart und -größe nicht modifiziert werden muß, sollte der Platzbedarf auszugebender Strings immer über *textwidth* und *textheight* berechnet werden.

Hinweis: Ein automatischer Umbruch wie im Textmodus findet nicht statt: Ausgaben, die über den rechten Rand des Bildschirms bzw. des aktuellen Zeichenfensters hinausgehen würden, schneidet *outtextxy* einfach ab.

outtextxy funktioniert nur im Grafik-, nicht aber im Textmodus.

Rückgabewert — Keiner.

Siehe auch — *gettextsettings*, *outtext*, *textheight*, *textwidth*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
```

```
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* output text at the center of the screen */
   /* Note: the C.P. doesn't get changed.     */
   outtextxy(midx, midy, "This is a test.");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

_OvrInitEms

Funktion

Initialisiert den Expansionsspeicher für die Nutzung durch die Overlay-Speicherverwaltung.

Syntax

```
#include <dos.h>
int cdecl far _OvrInitEms(unsigned emsHandle,
                          unsigned firstPage,
                          unsigned pages);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

_OvrInitEms prüft, ob Expansionsspeicher vorhanden ist, indem es einen EMS-Treiber sucht und von ihm Speicher reserviert.

Ist *emsHandle* gleich Null, reserviert die Overlay-Speicherverwaltung EMS-Seiten und verwendet sie zur Auslagerung von Daten.

Ist *emsHandle* nicht Null, muß es ein gültiges EMS-Handle

sein, das die Overlay-Speicherverwaltung zur Datenauslagerung verwenden kann. In diesem Fall können Sie mit *firstPage* die erste Seite innerhalb dieses Bereichs angeben, die zur Auslagerung verwendet werden soll.

In beiden Fällen legt ein *pages*-Argument ungleich Null die Höchstzahl der Seiten fest, die von der Overlay-Speicherverwaltung genutzt werden sollen.

Rückgabewert *_OvrInitEms* liefert 0 zurück, wenn die Overlay-Speicherverwaltung den Expansionsspeicher nutzen kann.

Siehe auch *_OvrInitExt*, *_ovrbuffer* (globale Variable)

Beispiel

```
#include <dos.h>

int main(void)
{
/*
   Ask overlay manager to check for expanded
   memory and allow it to use 16 pages
   (256K); available only in medium, large
   and huge memory models
*/
   _OvrInitEms (0, 0, 16);

   return 0;
}
```

_OvrInitExt

Funktion Initialisiert den Erweiterungsspeicher für die Nutzung durch die Overlay-Speicherverwaltung.

Syntax

```
#include <dos.h>
int cdecl far _OvrInitExt(unsigned long startAddress,
                          unsigned long length);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *_OvrInitExt* prüft, ob Erweiterungsspeicher vorhanden ist (d.h. die Funktion sucht mit den bekannten Methoden nach anderen Programmen, die den Erweiterungsspeicher benutzen), und reserviert Speicher.

Ist *startAddress* Null, bestimmt die Overlay-Speicherverwaltung die Startadresse und verwendet höchstens die Größe der Overlays. Ist *startAddress* ungleich Null, verwendet die Overlay-Speicherverwaltung den Erweiterungsspeicher oberhalb dieser Adresse.

In beiden Fällen legt ein *length*-Argument ungleich Null den Anteil am Erweiterungsspeicher fest, der maximal von der Overlay-Speicherverwaltung genutzt werden soll.

Rückgabewert *_OvrInitExt* liefert 0 zurück, wenn die Overlay-Speicherverwaltung den Erweiterungsspeicher nutzen kann.

Siehe auch *_OvrInitEms*, *_ovrbuffer* (globale Variable)

Beispiel

```
#include <dos.h>

int main(void)
{
/*
   Available only in medium, large and
   huge memory models
*/

/*
   Use the extended memory from the linear
   address 0x200000L (2MB), as much as
   necessary
*/
/* In the IDE, set Options|Applications to Dos Overlay
*/

   _OvrInitExt (0x200000L, 0);

   return 0;
}
```

parsfnm

Funktion Analysiert einen Dateinamen.

Syntax

```
#include <dos.h>
char *parsfnm(const char *cmdline, struct fcb *fcb,
              int opt);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *parsfnm* verwendet die DOS-Funktion 0x29, um einen Dateinamen zu analysieren und die Elemente einer Struktur des Typs *fcb* entsprechend zu setzen. Über den Parameter *cmdline* wird der zu analysierende String übergeben, **fcb* zeigt auf die zu setzende Struktur. Der Parameter *opt* bestimmt die Art der Analyse.

Weitere Informationen finden Sie in der Beschreibung des Systemaufrufs 0x29 in Ihren *DOS-Referenzhandbüchern*.

Rückgabewert Wenn die Analyse fehlerfrei ausgeführt werden konnte, dann liefert *parsfnm* einen Zeiger auf das erste Zeichen von *cmdline* zurück, das nicht mehr zum Dateinamen gehört. Im Falle eines Fehlers ist der Rückgabewert NULL.

Beispiel

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   char line[80];
   struct fcb blk;

   /* get file name */
   printf("Enter drive and file name "
          "(no path; e.g., a:file.dat)\n");
   gets(line);

   /* put file name in fcb */
   if (parsfnm(line, &blk, 1) == NULL)
      printf("Error in parsfm call\n");
   else
      printf("Drive #%d  Name: %11s\n",
             blk.fcb_drive, blk.fcb_name);

   return 0;
}
```

peek

Funktion Liest ein WORD an der Position *segment:offset*.

Syntax

```
#include <dos.h>
int peek(unsigned segment, unsigned offset);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung *peek* liest das WORD an der zwei Position *segment:offest*.

Die Funktion ist in dos.h als inline-Makro definiert – nur wenn Datei nicht aufgenommen wurde (bzw. nach einem entsprechenden #*undef peek*), erzeugt der Compiler »echte« Funktionsaufrufe.

Rückgabewert *peek* liefert das WORD an der durch *segment:offset* angegebenen Adresse zurück.

Siehe auch *harderr, peekb, poke*

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

int main(void)
{
   int value = 0;

   printf("The current status of your keyboard is:\n");
   value = peek(0x0040, 0x0017);
   if (value & 1)
      printf("Right shift on\n");
   else
      printf("Right shift off\n");

   if (value & 2)
      printf("Left shift on\n");
   else
      printf("Left shift off\n");

   if (value & 4)
      printf("Control key on\n");
   else
      printf("Control key off\n");

   if (value & 8)
      printf("Alt key on\n");
   else
      printf("Alt key off\n");

   if (value & 16)
      printf("Scroll lock on\n");
   else
      printf("Scroll lock off\n");

   if (value & 32)
      printf("Num lock on\n");
   else
      printf("Num lock off\n");

   if (value & 64)
      printf("Caps lock on\n");
   else
      printf("Caps lock off\n");

   return 0;
}
```

peekb

Funktion
Liest ein Byte an der Position *segment:offset*.

Syntax
```
#include <dos.h>
char peekb(unsigned segment, unsigned offset);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung
peekb liest das über *segment* und *offset* angegebene Byte.

Die Funktion ist in dos.h als inline-Makro definiert – nur wenn diese Datei nicht aufgenommen wurde (bzw. nach einem entsprechenden #*undef peekb*), erzeugt der Compiler »echte« Funktionsaufrufe.

Rückgabewert
peekb liefert den Inhalt des durch *segment:offset* adressierten Byte als **char** zurück.

Siehe auch
peek, pokeb

Beispiel
```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

int main(void)
{
   int value = 0;

   printf("The current status of your keyboard is:\n");
   value = peekb(0x0040, 0x0017);
   if (value & 1)
      printf("Right shift on\n");
   else
      printf("Right shift off\n");

   if (value & 2)
      printf("Left shift on\n");
   else
      printf("Left shift off\n");

   if (value & 4)
      printf("Control key on\n");
   else
      printf("Control key off\n");

   if (value & 8)
      printf("Alt key on\n");
   else
      printf("Alt key off\n");
```

```
   if (value & 16)
      printf("Scroll lock on\n");
   else
      printf("Scroll lock off\n");

   if (value & 32)
      printf("Num lock on\n");
   else
      printf("Num lock off\n");

   if (value & 64)
      printf("Caps lock on\n");
   else
      printf("Caps lock off\n");

   return 0;
}
```

perror

Funktion Gibt eine Systemfehlermeldung zu *stderr* aus.

Syntax
```
#include <stdio.h>
void perror(const char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung *perror* kann nach einem Fehler aufgerufen werden und gibt zwei Strings zu *stderr* aus (normalerweise der Bildschirm):

- den String *s*;
- danach einen Doppelpunkt und die Fehlermeldung, die dem aktuellen Wert von *errno* entspricht;
- einen Zeilenvorschub.

Das Array *sys_errlist* enthält sämtliche Fehlermeldungen des Systems in der Reihenfolge aufsteigender Fehlernummern. *perror* verwendet den aktuellen Wert der globalen Variablen *errno* zur Adressierung.

Die Fehlermeldungen enthalten kein Zeichen für einen Zeilenvorschub. Die globale Variable *sys_nerr* enthält die Anzahl der Einträge im Array *sys_errlist*.

Weitere Details zu *errno*, *sys_errlist* und *sys_nerr* finden Sie in Kapitel 3 »Globale Variablen«.

Rückgabewert Keiner.

Siehe auch *clearerr*, *eof*, *_strerror*, *strerror*

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *fp;

   fp = fopen("perror.dat", "r");
   if (!fp)
      perror("Unable to open file for reading");
   return 0;
}
```

pieslice

Funktion Zeichnet ein ausgefülltes »Kuchenstück«.

Syntax

```
#include <graphics.h>
void far pieslice(int x, int y, int stangle,
                  int endangle, int radius);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *pieslice* zeichnet in der aktuellen Zeichenfarbe ein Kreisbogenstück mit dem Mittelpunkt (*x*,*y*), wobei *stangle* den Startpunkt und *endangle* den Endpunkt des mit *radius* gezogenen Bogens festlegen.

Die Endpunkte des Kreisbogens werden mit dem Kreismittelpunkt verbunden, danach füllt *pieslice* das entstandene Kuchenstück mit dem durch *setfillpattern* bzw. *setfillstyle* festgelegten Muster aus.

Die Angabe beider Winkel erfolgt in Grad, wobei entgegen dem Uhrzeigersinn gezählt wird (0 Grad = horizontal rechts des Mittelpunktes, 90 Grad = senkrecht über dem Mittelpunkt usw).

Hinweis: Wenn Ihr System mit einem CGA- oder Monochromadapter ausgestattet ist, werden einige der Beispiele zur Verwendung der Grafikfunktionen nicht einwandfrei funktionieren. Sie sollten dann den Wert 1 (eins) statt der symbolischen Farbkonstanten an die Funktionen übergeben, die die Füll- oder Zeichenfarbe setzen (zum Beispiel *setcolor*, *setfillstyle* oder *setlinestyle*). Verwenden Sie das zweite Beispiel unter *arc*.

Rückgabewert Keiner.

Siehe auch *fillellipse*, *fill_patterns* (Aufzählungstyp), *graphresult*, *sector*, *setfillstyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 45, endangle = 135, radius = 100;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* set fill style and draw a pie slice */
   setfillstyle(EMPTY_FILL, getmaxcolor());
   pieslice(midx, midy, stangle, endangle, radius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

poke

Funktion

Schreibt einen **int**-Wert an die durch *segment:offset* angegebenen Speicherposition.

Syntax

```
#include <dos.h>
void poke(unsigned segment, unsigned offset,
          int value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *poke* speichert den durch *value* angegebenen Integer, wobei *segment:offset* die Position angibt.

Die Funktion ist in dos.h als inline-Makro definiert. Nur wenn diese Datei nicht mit aufgenommen wurde (bzw. nach einem entsprechenden #*undef poke*), verwendet der Compiler »echte« Funktionsaufrufe.

Rückgabewert Keiner.

Siehe auch *harderr, peek, pokeb*

Beispiel

```
#include <dos.h>
#include <conio.h>

int main(void)
{
   clrscr();
   cprintf("Make sure the scroll lock key is off "
           "and press any key\r\n");
   getch();
   poke(0x0000,0x0417,16);
   cprintf("The scroll lock is now on\r\n");
   return 0;
}
```

pokeb

Funktion Speichert einen Byte-Wert an der Position *segment:offset*.

Syntax

```
#include <dos.h>
void pokeb(unsigned segment, unsigned offset,
           char value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *pokeb* schreibt den Byte-Wert *value* an die Speicherposition mit der Adresse *segment:offset*.

Die Funktion ist in dos.h als inline-Makro definiert. Nur wenn diese Datei nicht mit aufgenommen wurde (bzw. nach einem entsprechenden #*undef pokeb*), verwendet der Compiler »echte« Funktionsaufrufe.

Rückgabewert Keiner.

Siehe auch *peekb, poke*

Beispiel

```
#include <dos.h>
#include <conio.h>

int main(void)
{
```

```
   clrscr();
   cprintf("Make sure the scroll lock key is off "
           "and press any key\r\n");
   getch();
   pokeb(0x0000,0x0417,16);
   cprintf("The scroll lock is now on\r\n");
   return 0;
}
```

polar

Funktion

Liefert die zu einem gegebenen Radius und Winkel gehörige komplexe Zahl.

Syntax

```
#include <complex.h>
complex polar(double mag, double angle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung

polar (*mag, angle*) ist äquivalent zu *complex* (*mag* * *cos* (*angle*), *mag* * *sin* (*angle*)).

Rückgabewert

polar liefert die komplexe Zahl zum gegebenen Radius *mag* (absoluter Wert) und dem mit *angle* gegebenen Winkel zurück.

Siehe auch

arg, complex, norm

Beispiel

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary real part = "
        << imag(z) << "\n";
   cout << "z has complex conjugate = "
        << conj(z) << "\n";

   double mag = sqrt(norm(z));
   double ang = arg(z);
   cout << "The polar form of z is:\n";
   cout << "  magnitude = " << mag << "\n";
   cout << "  angle (in radians) = " << ang << "\n";
   cout << "Reconstructing z from its "
           "polar form gives:\n";
```

```
   cout << "   z = " << polar(mag,ang) << "\n";
   return 0;
}
```

poly, polyl

Funktion

Erzeugt ein Polynom aus den angegebenen Argumenten.

Syntax

```
#include <math.h>
double poly(double x, int degree, double coeffs[]);
long double polyl(long double x, int degree,
                  long double coeffs[]);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x			*poly*
x		x			*polyl*

Beschreibung

poly bildet ein Polynom mit dem Grad *degree* über den Wert von *x* zusammen mit den Koeffizienten *coeffs*[*degree*], *coeffs*[*degree-1*], ..., *coeffs*[*0*].

Ein Aufruf mit dem Wert 3 für *degree* berechnet beispielsweise den folgenden Ausdruck:

$$coeffs[3]x^3 + coeffs[2]x^2 + coeffs[1]x + coeffs[0]$$

polyl ist die **long double** Version. Sie erwartet Argumente vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Rückgabewert

Zurückgeliefert wird das Ergebnis der Polynom-Berechnung für den gegebenen Wert *x*.

Beispiel

```
#include <stdio.h>
#include <math.h>

/* polynomial:  x**3 - 2x**2 + 5x - 1 */

int main(void)
{
   double array[] = { -1.0, 5.0, -2.0, 1.0
};
   double result;

   result = poly(2.0, 3, array);
   printf("The polynomial: x**3 - 2.0x**2 + 5x - 1 "
          "at 2.0 is %lf\n",
           result);
   return 0;
}
```

pow, powl

Funktion Berechnet x^y.

Syntax reelle Versionen:

```
#include <math.h>
double pow(double x, double y);
long double powl(long double x, long double y);
```

komplexe Versionen:

```
#include <complex.h>
complex pow(complex    x, complex   y);
complex pow(complex    x, double    y);
complex pow(double     x, complex   y);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*powl*
x	x	x	x		*reelles pow*
x		x		x	*komplexes pow*

Beschreibung *pow* liefert das Ergebnis der Berechnung x^y zurück.

powl ist die **long double** Version. Sie erwartet Argumente vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Die komplexe Variante von *pow* wird durch folgende Gleichung definiert:

$$pow(base, expon) = exp(expon\ log(base))$$

Rückgabewert Bei fehlerfreier Ausführung liefern *pow* und *powl* das Ergebnis der Berechnung x^y zurück.

Falls sich die Argumente nicht innerhalb der zulässigen Grenzen halten, werden die folgenden Fehlerwerte erzeugt:

Bei einem Überlauf liefert *pow* HUGE_VAL und *powl* _LHUGE_VAL. Die globale Variable *errno* wird in diesen Fällen auf folgenden Wert gesetzt:

ERANGE Ergebnis außerhalb des Rechenbereichs

Unzulässige Argumente (falls *y* keine ganze Zahl bzw. die reelle Zahl $x <= 0$ ist) setzen *errno* auf den Wert

EDOM Bereichsfehler

Bei *x* und *y* gleich 0 liefern *pow* und *powl* den Wert 1.

Die Behandlung von Fehlern kann für diese Funktionen über die Routinen *matherr* und *_matherrl* modifiziert werden.

Siehe auch *complex, exp, pow10, sqrt*

Beispiel

```
#include <math.h>
#include <stdio.h>
```

```
int main(void)
{
   double x = 2.0, y = 3.0;

   printf("%lf raised to %lf is %lf\n",x,y, pow(x, y));
   return 0;
}
```

pow10, pow10l

Funktion Berechnet 10^{p}.

Syntax
```
#include <math.h>
double pow10(int p);
long double pow10l(int p);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x			*pow10*
x		x			*pow10l*

Beschreibung *pow10* liefert das Ergebnis der Berechnung 10^{p} zurück und ist das Gegenstück zu *log10*.

Rückgabewert *pow10* liefert das Ergebnis der Berechnung von 10^{p} zurück. Das Resultat wird mit der Genauigkeit eines **long double** berechnet. Alle Argumente sind zulässig, können jedoch in manchen Fällen zu einem Überlauf oder Unterlauf führen.

pow10l ist die **long double** Version und liefert ein Ergebnis vom Typ **long double** zurück.

Siehe auch *exp, pow*

Beispiel
```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double p = 3.0;

   printf("Ten raised to %lf is %lf\n", p, pow10(p));
   return 0;
}
```

printf

Funktion Gibt formatiert zu *stdout* aus.

Syntax
```
#include <stdio.h>
int printf(const char *format[argument, ...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung

Alle *...printf*-Funktionen geben Daten in formatierter Form an *stdout* aus und erwarten zumindest einen Formatstring (Parameter *format*), über den die Anzahl der Ausgaben und ihr Format festgelegt wird. Die Anzahl von Formatangaben und Adressen muß der Anzahl der Argumente genau entsprechen.

Der Formatstring

Der Formatstring ist in jeder Variante von *printf* obligatorisch – über ihn wird die Art der Konvertierung, die Formatierung und die Anzahl der Parameter festgelegt.

Parameter, die durch den Formatstring nicht erfaßt (das heißt überzählig) sind, werden ignoriert. Wenn *printf* aufgrund des Formatstrings dagegen mehr Argumente erwartet, als angegeben sind, ergeben sich bestenfalls merkwürdige, meist aber katastrophale Resultate.

Der Formatstring ist ein nullterminierter String, der zwei Arten von Informationen enthält:

- »normale Zeichen«, die direkt ausgegeben werden;
- Konvertierungsanweisungen, die die Auswertung korrespondierender Argumente und ihre Formatierung bestimmen.

Konvertierungsanweisungen

Konvertierungsanweisungen für *printf* lassen sich durch das folgende Syntaxdiagramm darstellen:

% [Flags] [Breite] [.Präzision] [F|N|h|l|L] Typ

Jede Anweisung beginnt mit einem Prozentzeichen (%). Auf dieses Zeichen folgen (in der angegebenen Reihenfolge):

[*Flags*] – Eine (optionale) Zeichenfolge, über die numerische Vorzeichen, Dezimalpunkte, führende und folgende Nullen, oktale und hexadezimale Präfixe sowie links- und rechtsbündige Ausgabe festgelegt werden.

[*Breite*] – Eine (optionale) Angabe über die minimal auszugebende Zeichenzahl (*printf* füllt nötigenfalls mit Leerzeichen oder Nullen auf).

[*.Präzision*] – Eine (optionale) Angabe, wieviele Zeichen maximal ausgegeben werden sollen (bei Integern: die Minimalzahl von Ziffern).

[F|N|h|l|L] – Eine (optionale) Angabe der Größe des Arguments (*F* = **far**-Zeiger, *N* = **near**-Zeiger, *h* = **short int** , *l* = **long** , *L* = **long double**). Nur notwendig, wenn die Größe des Para-

meters von der »Standardgröße« abweicht (**far**-Zeiger in kleinen Speichermodellen usw.).

Typ – die Angabe des Typs des konvertierten Zeichens.

Die folgende Tabelle enthält die Typen konvertierter Zeichen der *...printf*-Funktionen. Zusätzlich ist der Typ des entsprechenden Eingabe-Arguments und das Format der Ausgabe angegeben. Die Informationen in der Tabelle gelten nur, wenn keine Flags oder sonstigen Bezeichner und Modifizierer (s.o) angegeben wurden. Deren Auswirkung wird in den folgenden Tabellen berücksichtigt.

Typ	Parameter	Ausgabe
Numerische Werte		
d	Integer	**signed int**(dezimal).
i	Integer	**signed int**(dezimal)
o	Integer	**unsigned int**(oktal)
u	Integer	**unsigned int**(dezimal)
x	Integer	**unsigned int**(hexadezimal), Buchstaben a bis f
X	Integer	**unsigned int**(hexadezimal), Buchstaben A bis F. Die Formate *x* und *X* unterscheiden sich nur durch die Verwendung von Klein- bzw. Großbuchstaben.
f	Gleitkomma	Vorzeichenbehafteter Wert der Form [-]*dddd.dddd* Die Anzahl der Nachkommastellen wird durch Präzision festgelegt (wenn dieser Zusatz angegeben ist).
e	Gleitkomma	Vorzeichenbehafteter Wert der Form [-]*d.dddd* oder [+/-]*ddd* Hier steht grundsätzlich eine Ziffer vor dem Dezimalpunkt, die Anzahl der Nachkommastellen wird durch Präzision festgelegt, der Exponent umfaßt mindestens zwei Ziffern (nötigenfalls mit führenden Nullen).
g	Gleitkomma	Vorzeichenbehafteter Wert im *e*- oder *f*-Format. *Präzision* gibt die Anzahl signifikanter Ziffern an. Abschließende Nullen und der Dezimalpunkt werden nur ausgegeben, wenn es notwendig ist. Das *e*-Format wird nur dann verwendet, wenn der aus der Konvertierung resultierende Exponent größer als die Präzision oder kleiner als -4 ist.
E	Gleitkomma	Dasselbe Format wie *e*, aber mit dem Zeichen *E* vor dem Exponenten;

Typ	Parameter	Ausgabe
G	Gleitkomma	Dasselbe Format wie *g*, aber mit dem Zeichen *E* vor dem Exponenten, wenn das Format *e* verwendet wird.
Zeichen		
c	Zeichen	Einzelnes Zeichen.
s	String	Ausgabe bis zu einem Nullzeichen oder dem Erreichen der durch *Präzision* vorgegebenen Zeichenzahl.
%	(nichts)	Ausgabe des Zeichens %.
Zeiger		
n	***int**	Speichert an der Position, auf die das Eingabeargument zeigt, die Anzahl der bis jetzt ausgegebenen Zeichen.
p	Zeiger	Gibt das Eingabeargument als Zeiger in hexadezimaler Form aus (**near**-Zeiger als YYYY, **far**-Zeiger als XXXX:YYYY).

Hinweis: Unendlich große positive oder negative Gleitkommazahlen werden als *+INF* bzw. als *-INF* ausgegeben, eine »IEEE-Nichtzahl« als *+NAN* oder *-NAN*.

Flags

printf erkennt vier verschiedene Zeichen als Flags an: Minus (-), Plus (+), Doppelkreuz (#) und Leerzeichen( ). Flag-Zeichen können in beliebiger Folge und Kombination erscheinen und haben die folgenden Wirkungen:

Flag	Bedeutung
-	Linksbündige Ausgabe – nötigenfalls werden nach der Ausgabe Leerzeichen angefügt. Wenn dieses Flag nicht angegeben ist, arbeitet *printf* rechtsbündig, das heißt füllt links mit Leerzeichen oder Nullen auf.
+	Numerische Ausgabe mit Vorzeichen – positiven Werten wird ein Pluszeichen vorangestellt. Ist dieses Flag nicht angegeben, stellt *printf* nur bei negativen Werten ein Minuszeichen voran, bei positiven Werten überhaupt nichts.
Leerzeichen	Positive Werte werden mit einem führenden Leerzeichen ausgegeben. Wenn weder dieses Flag noch ein + angegeben ist, dann stellt *printf* positiven Werten überhaupt nichts voran.
#	Konvertierung des Arguments in »alternativer Form« (siehe folgende Tabelle).

Hinweis: Wenn sowohl ein Leerzeichen als auch ein Pluszeichen angegeben sind, wird das Leerzeichen von *printf* ignoriert.

Alternative Formen
Die Verwendung des Flags # zusammen mit einer Typ-Angabe hat auf das zu konvertierende Argument (arg) folgende Wirkungen:

Format ausgabe	Wirkung von # auf *arg*
c,s,d,i,u	Keine Wirkung.
0	Eine 0 wird vorangestellt, wenn *arg* ungleich 0 ist.
x,X	Voranstellen von »0x« bzw. »0X«.
e,E,f	Es wird immer ein Dezimalpunkt ausgegeben (normalerweise ist das bei diesen Formaten nur dann der Fall, wenn der Wert Stellen nach dem Komma enthält).
g,G	Wie bei *e* und *E* – zusätzlich werden abschließende Nullen nicht unterdrückt.

Breite
Diese Angabe legt die minimale Zahl auszugebender Zeichen fest und kann in zwei Formen erfolgen: entweder direkt innerhalb des Formatstrings oder indirekt über ein Sternchen (*). Bei der Verwendung von * erwartet *printf* ein zusätzliches Argument, das in der Liste direkt *vor* dem auszugebenden Wert stehen und den Typ **int** haben muß.

Wenn eine Ausgabe weniger Zeichen erzeugt als durch *Breite* angegeben ist, dann fügt *printf* (abhängig von den *Flags*) führende/abschließende Leerzeichen hinzu – erzeugt die Ausgabe dagegen mehr Zeichen als durch *Breite* angegeben, dann wird *Breite* einfach ignoriert, Ausgaben werden durch *Breite* in keinem Fall begrenzt oder abgeschnitten.

Angabe	Wirkung
n	(Dezimalzahl) – *printf* gibt mindestens *n* Zeichen aus und stellt nötigenfalls Leerzeichen voran (Standard) bzw. fügt abschließende Leerzeichen an (Flag -).
0n	(Dezimalzahl mit vorangestellter Null) – *printf* gibt mindestens *n* Zeichen aus und stellt nötigenfalls eine entsprechende Anzahl führender Nullen voran.
*	(Sternchen) – Die Angabe für Breite steht in der Parameterliste direkt vor dem auszugebenden Wert.

Präzision
Diese Angabe wird immer mit einem Dezimalpunkt (.) eingeleitet und so von einer eventuell vorangestellten *Breite* getrennt. Wie bei *Breite* kann entweder eine direkte Angabe innerhalb des Formatstrings oder ein Sternchen (*) verwendet

werden. Im letzteren Fall erwartet *printf* einen zusätzlichen Parameter des Typs **int**, der direkt *vor* dem auszugebenden Wert steht.

Wenn sowohl *Breite* als auch *Präzision* über ein Sternchen angegeben werden, muß die Parameterliste wie folgt aussehen:

```
("%<Flags>*.*<Typ>", Breite, Präzision, Wert)
```

Mögliche Formen der Angabe für *Präzision* und ihre Wirkungen sind:

Angabe	Wirkung
keine:	1 – für *d*, *i*, *o*, *u*, *x* und *X*. 6 – für *e*, *E* und *f*. Sämtliche signifikanten Stellen für *g* und *G*. Ausgabe bis zum ersten Nullzeichen für *s*. Keine Auswirkungen für *c*.
.0	Standardvorgaben für *d*, *i*, *o*, *u*, *x* und *X*, keine Ausgabe des Dezimalpunkts für *e*, *E* und *f*.
.n	Ausgabe von *n* Zeichen bzw. *n* signifikanten Dezimalstellen. Falls der Ausgabewert mehr als *n* Zeichen umfaßt, wird die Ausgabe unter Umständen verkürzt oder gerundet.
*	Die Angabe für *Präzision* ist in der Parameterliste enthalten und muß dort als **int** vor dem auszugebenden Wert stehen.

Hinweis: Wenn für *Präzision* explizit Null angegeben wird, es sich um ein Integerformat handelt (*d*, *i*, *o*, *u* oder *x*) und der auszugebende Wert 0 ist, wird für dieses Feld kein numerisches Zeichen ausgegeben (d.h. das Feld ist leer).

Format-ausgabe	Wirkung von .n auf die Konvertierung
d, *i*, *o*, *u*, *x*, *X*	Nötigenfalls stellt *printf* führende Nullen voran – falls der auszugebende Wert mehr Stellen umfaßt als durch *n* angegeben, wird der ausgegebene Wert nicht abgeschnitten.
e, *E*, *f*	*n* legt die Anzahl der Stellen nach dem Komma fest – die letzte Stelle wird entsprechend gerundet.
g, *G*	*n* legt die Anzahl signifikanter Dezimalstellen fest (inklusive der Stellen vor dem Komma).
c	Die Angabe von *n* hat keine Wirkung.
s	Es werden höchstens *n* Zeichen ausgegeben.

F | *N* | *h* | *l* | *L* – **explizite Größenangaben**
Die Größenangaben *F* (**far**-Zeiger) und *N* (**near**-Zeiger) legen eine (vom aktuellen Speichermodell abweichende) Zeigergrö-

ße fest und werden nur für die Ausgaben von Zeigerwerten verwendet.

Die Größenangaben *h*, *l* und *L* sind nur auf numerische Argumente anwendbar – für die Formate *c*, *s*, *p* und *n* haben sie keine Wirkung.

Angabe	Wirkung auf die Interpretation von *arg*
F	*arg* wird als **far**-Zeiger gelesen und im Format XXXX:YYYY ausgegeben.
N	*arg* wird als **near**-Zeiger gelesen und im Format YYYY ausgegeben; *N* kann im Modell *huge* nicht verwendet werden.
h	Das Argument wird als **short int** interpretiert (Formate *d*, *i*, *o*, *u*, *x* und *X*).
l	Das Argument wird als **long int** (Formate *d*, *i*, *o*, *u*, *x* und X) bzw. als **double** interpretiert (Formate *e*, *E*, *f*, *g* und *G*).
L	Das Argument wird als **long double** interpretiert (Formate *e*, *E*, *f*, *g* und *G*).

Rückgabewert *printf* liefert bei fehlerfreier Ausführung die Anzahl der ausgegebenen Bytes, im Falle eines Fehlers EOF zurück.

Siehe auch *cprintf*, *ecvt*, *fprintf*, *fread*, *fscanf*, *putc*, *puts*, *putw*, *scanf*, *sprintf*, *vprintf*, *vsprintf*

Beispiel

```
#include <stdio.h>
#include <string.h>

#define I 555
#define R 5.5

int main(void)
{
   int i,j,k,l;
   char buf[7];
   char *prefix = buf;
   char tp[20];
   printf("prefix  6d      6o      8x        "
          "10.2e        "
          "10.2f\n");
   strcpy(prefix,"%");
   for (i = 0; i < 2; i++)
   {
      for (j = 0; j < 2; j++)
         for (k = 0; k < 2; k++)
            for (l = 0; l < 2; l++)
            {
```

```
                        if (i==0)  strcat(prefix,"-");
                        if (j==0)  strcat(prefix,"+");
                        if (k==0)  strcat(prefix,"#");
                        if (l==0)  strcat(prefix,"0");
                        printf("%5s |",prefix);
                        strcpy(tp,prefix);
                        strcat(tp,"6d |");
                        printf(tp,I);
                        strcpy(tp,"");
                        strcpy(tp,prefix);
                        strcat(tp,"6o |");
                        printf(tp,I);
                        strcpy(tp,"");
                        strcpy(tp,prefix);
                        strcat(tp,"8x |");
                        printf(tp,I);
                        strcpy(tp,"");
                        strcpy(tp,prefix);
                        strcat(tp,"10.2e |");
                        printf(tp,R);
                        strcpy(tp,prefix);
                        strcat(tp,"10.2f |");
                        printf(tp,R);
                        printf("  \n");
                        strcpy(prefix,"%");
                    }
                }
            return 0;
        }
```

Programmausgabe:

```
prefix   6d     6o       8x        10.2e       10.2f
 %-+#0|+555   |01053 |0x22b    |+5.50e+00  |+5.50     |
  %-+#|+555   |01053 |0x22b    |+5.50e+00  |+5.50     |
  %-+0|+555   |1053  |22b      |+5.50e+00  |+5.50     |
   %-+|+555   |1053  |22b      |+5.50e+00  |+5.50     |
  %-#0|555    |01053 |0x22b    |5.50e+00   |5.50      |
   %-+|555    |01053 |0x22b    |5.50e+00   |5.50      |
   %-0|555    |1053  |22b      |5.50e+00   |5.50      |
    %-|555    |1053  |22b      |5.50e+00   |5.50      |
  %+#0|+00555 |001053|0x00022b |+05.50e+00 |+000005.50|
   %+#|  +555 | 01053|   0x22b | +5.50e+00 |     +5.50|
   %+0|+00555 |001053|0000022b |+05.50e+00 |+000005.50|
    %+|  +555 |  1053|     22b | +5.50e+00 |     +5.50|
   %#0|000555 |001053|0x00022b |005.50e+00 |0000005.50|
    %#|   555 | 01053|   0x22b |  5.50e+00 |      5.50|
    %0|000555 |001053|0000022b |005.50e+00 |0000005.50|
     % |   555 |  1053|     22b |  5.50e+00 |      5.50|
```

putc

Funktion Gibt ein Zeichen zu einem Stream aus.

Syntax
```
#include <stdio.h>
int putc(int c, FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung Das Makro *putc* gibt das Zeichen *c* zu dem durch *stream* angegebenen Stream aus.

Rückgabewert *putc* liefert bei fehlerfreier Ausführung das übergebene Zeichen *c*, im Falle eines Fehlers den Wert EOF zurück.

Siehe auch *fprintf*, *fputc*, *fputch*, *fputchar*, *fputs*, *fwrite*, *getc*, *getchar*, *printf*, *putch*, *putchar*, *putw*, *vprintf*

Beispiel
```
#include <stdio.h>

int main(void)
{
   char msg[] = "Hello world\n";
   int i = 0;

   while (msg[i])
      putc(msg[i++], stdout);
   return 0;
}
```

putch

Funktion Gibt ein Zeichen auf den Bildschirm aus.

Syntax
```
#include <conio.h>
int putch(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *putch* gibt das Zeichen *c* auf den Bildschirm in das aktuelle Textfenster aus. Eine Übersetzung von Zeilenvorschubzeichen (*\n*) in CR/LF-Paare findet nicht statt. *putch* funktioniert nur im Textmodus.

Abhängig vom Wert der globalen Variablen *directvideo* wird der String entweder direkt in den Bildschirmspeicher geschrieben oder über den Umweg eines BIOS-Aufrufs ausgegeben.

Rückgabewert *putch* liefert bei fehlerfreier Ausführung das übergebene Zeichen *c*, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch *cprintf, cputs, getch, getche, putc, putchar*

Beispiel

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
   char ch = 0;

   printf("Input a string:");
   while ((ch != '\r'))
   {
      ch = getch();
      putch(ch);
   }
   return 0;
}
```

putchar

Funktion *putchar* gibt ein Zeichen zu *stdout* aus.

Syntax

```
#include <stdio.h>
int putchar(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *putchar* ist ein Makro und als *putc(c, stdout)* definiert, gibt also ein Zeichen zu *stdout* aus.

Rückgabewert *putchar* liefert bei fehlerfreier Ausführung das übergebene Zeichen *c*, im Fall eines Fehlers den Wert EOF zurück.

Siehe auch *fputchar, getc, getchar, printf, putc, putch, puts, putw, vprintf*

Beispiel

```
#include <stdio.h>

/* define some box-drawing characters */
#define LEFT_TOP  0xDA
#define RIGHT_TOP 0xBF
#define HORIZ     0xC4
#define VERT      0xB3
#define LEFT_BOT  0xC0
#define RIGHT_BOT 0xD9

int main(void)
{
   char i, j;
```

```
   /* draw the top of the box */
   putchar(LEFT_TOP);
   for (i=0; i<10; i++)
      putchar(HORIZ);
   putchar(RIGHT_TOP);
   putchar('\n');

   /* draw the middle */
   for (i=0; i<4; i++)
   {
      putchar(VERT);
      for (j=0; j<10; j++)
         putchar(' ');
      putchar(VERT);
      putchar('\n');
   }

   /* draw the bottom */
   putchar(LEFT_BOT);
   for (i=0; i<10; i++)
      putchar(HORIZ);
   putchar(RIGHT_BOT);
   putchar('\n');
   return 0;
}
```

putenv

Funktion

Fügt der Betriebssystemtabelle *Environment* einen String hinzu (bzw. löscht/verändert einen String).

Syntax

```
#include <stdlib.h>
int putenv(const char *name);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

putenv erwartet einen String *name* als Argument (zum Beispiel *putenv("PATH=C:\\BC");*) und fügt ihn zur *Environment*-Tabelle des aktuellen Prozesses hinzu.

Mit *putenv* kann auch ein vorhandener String geändert oder gelöscht werden. Eine Angabe der Form *name=*(das heißt ohne *Zeichenfolge*) sucht den String *name* und löscht ihn vollständig.

putenv kann nur das Environment des laufenden Programmes verändern. Nach dem Programmende ist wieder die ursprüngliche *Environment*-Tabelle gültig.

Es dürfen nur statische oder globale Strings an *putenv* übergeben werden. Wird ein lokaler oder dynamischer String an *putenv* übergeben und benutzt, nachdem der Speicher des

Strings freigegeben wurde, so kann dies zu unerwünschten Ergebnissen führen.

Rückgabewert *putenv* liefert bei fehlerfreier Ausführung den Wert 0, andernfalls -1 zurück.

Siehe auch *getenv*

Beispiel
```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char *path, *ptr;
   int i = 0;

   /* get the current path environment */
   ptr = getenv("PATH");

   /* set up new path */
   path = (char *) malloc(strlen(ptr)+15);
   strcpy(path,"PATH=");
   strcat(path,ptr);
   strcat(path,";c:\\temp");

   /* replace the current path and display current
      environment */
   putenv(path);
   while (environ[i])
       printf("%s\n",environ[i++]);

   return 0;
}
```

putimage

Funktion Kopiert den Inhalt einer Puffervariablen bitweise in einen rechteckigen Bildausschnitt.

Syntax
```
#include <graphics.h>
void far putimage(int left, int top, void far *bitmap,
                  int op);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *putimage* gibt den zuvor mit *getimage* gespeicherten Bildausschnitt auf den Bildschirm aus, wobei (*left*,*top*) die obere linke

Ecke des »Zielbereichs« festlegt. *bitmap* zeigt auf den Speicherbereich, in dem der Bildausschnitt gespeichert ist.

Der Parameter *op* steuert, wie die Farbe jedes Pixels im »Zielbereich« basierend auf der Farbe des bereits dort vorhandenen Pixels und der des Pixels im Speicher (Quellbereich) berechnet wird.

Für den Parameter *op* sind in graphics.h die folgenden Namen für Operatoren definiert:

Name	Wert	Beschreibung
COPY_PUT	0	Kopie (Überschreiben des »Zielbereichs«)
XOR_PUT	1	Exklusives-ODER-Operation mit dem »Zielbereich«
OR_PUT	2	ODER-Operation
AND_PUT	3	UND-Operation
NOT_PUT	4	Quellbereich invertieren

Rückgabewert Keiner.

Siehe auch *getimage*, *imagesize*, *putpixel*, *setvisualpage*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
```

```
   maxx = getmaxx();
   x = 0;
   y = getmaxy() / 2;

   /* draw the image to be grabbed */
   draw_arrow(x, y);

   /* calculate the size of the image */
   size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
                    y+ARROW_SIZE);

   /* allocate memory to hold the image */
   arrow = malloc(size);

   /* grab the image */
   getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
            y+ARROW_SIZE, arrow);

   /* repeat until a key is pressed */
   while (!kbhit())
   {
      /* erase old image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);

      x += ARROW_SIZE;
      if (x >= maxx)
          x = 0;

      /* plot new image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
   }

   /* clean up */
   free(arrow);
   closegraph();
   return 0;
}

void draw_arrow(int x, int y)
{
   /* draw an arrow on the screen */
   moveto(x, y);
   linerel(4*ARROW_SIZE, 0);
   linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
   linerel(0, 2*ARROW_SIZE);
   linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
```

putpixel

Funktion Zeichnet einen Punkt auf den angegebenen Koordinaten.

Syntax
```
#include <graphics.h>
void far putpixel(int x, int y, int color);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *putpixel* zeichnet einen Punkt mit der Farbe *color* auf dem Punkt (*x*,*y*).

Rückgabewert Keiner.

Siehe auch *getpixel, putimage*

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME  100  /* in milliseconds */

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int i, x, y, color, maxx, maxy, maxcolor, seed;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   maxx = getmaxx() + 1;
   maxy = getmaxy() + 1;
   maxcolor = getmaxcolor() + 1;
```

```
   while (!kbhit())
   {
      /* seed the random number generator */
      seed = random(32767);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++)
      {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         putpixel(x, y, color);
      }

      delay(DELAY_TIME);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++)
      {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         if (color == getpixel(x, y))
            putpixel(x, y, 0);
      }
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

puts

Funktion — Gibt eine Zeichenfolge zu *stdout* aus.

Syntax

```
#include <stdio.h>
int puts(const char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung — *puts* schreibt den nullterminierten String *s* zeichenweise zu *stdout* und beendet die Ausgabe mit einem Zeilenvorschubzeichen.

Rückgabewert — Bei fehlerfreier Ausführung liefert die Funktion einen Wert größer Null zurück. Im Falle eines Fehlers ist der Rückgabewert EOF.

Siehe auch *cputs, fputs, gets, printf, putchar*

Beispiel

```
#include <stdio.h>

int main(void)
{
   char string[] ="This is an example output string\n";

   puts(string);
   return 0;
}
```

puttext

Funktion Kopiert den Inhalt einer Puffervariablen in einen rechteckigen Ausschnitt des Textbildschirms.

Syntax

```
#include <conio.h>
int puttext(int left, int top, int right, int bottom,
            void *source);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *puttext* schreibt den Inhalt des durch *source* bezeichneten Speicherbereichs in einen rechteckigen Ausschnitt des Textbildschirms. Der Ausschnitt wird durch die Parameter *left*, *top*, *right* und *bottom* festgelegt.

Die Funktion arbeitet mit absoluten Bildschirmkoordinaten. Die obere linke Ecke des Textbildschirms hat die Koordinaten (1,1).

puttext plaziert den Inhalt des angegebenen Speicherbereichs in sequentieller Reihenfolge in das definierte Rechteck von links nach rechts und von oben nach unten. *puttext* funktioniert nur im Textmodus und schreibt direkt in den Bildschirmspeicher.

Rückgabewert Bei fehlerfreier Ausführung wird ein Wert ungleich Null zurückgeliefert. Ein Rückgabewert von 0 zeigt einen Fehler an (wie zum Beispiel Koordinaten außerhalb des Bildschirms).

Siehe auch *gettext, movetext, window*

Beispiel

```
#include <conio.h>

int main(void)
{
   char buffer[512];

   /* put some text to the console */
   clrscr();
   gotoxy(20, 12);
```

```
   cprintf("This is a test.  "
           "Press any key to continue ...");
   getch();

   /* grab screen contents */
   gettext(20, 12, 36, 21,buffer);
   clrscr();

   /* put selected characters back to the screen */
   gotoxy(20, 12);
   puttext(20, 12, 36, 21, buffer);
   getch();

   return 0;
}
```

putw

Funktion — Gibt einen Integer zu einem Stream aus.

Syntax

```
#include <stdio.h>
int putw(int w, FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *putw* gibt den Integer *w* zu dem Stream *stream* aus. Eine spezielle Anordnung in der Datei wird von der Funktion weder verursacht noch erwartet.

Rückgabewert — *putw* liefert bei fehlerfreier Ausführung den übergebenen Integer *w*, im Falle eines Fehlers den Wert EOF (-1) zurück. Da dieser Wert ein legaler Integer ist, sollte für *putw* ein expliziter Test mit *ferror* durchgeführt werden.

Siehe auch — *getw*, *printf*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>

#define FNAME "test.$$$"

int main(void)
{
   FILE *fp;
   int word;

   /* place the word in a file */
   fp = fopen(FNAME, "wb");
   if (fp == NULL)
   {
```

```
        printf("Error opening file %s\n", FNAME);
        exit(1);
    }

    word = 94;
    putw(word,fp);
    if (ferror(fp))
        printf("Error writing to file\n");
    else
        printf("Successful write\n");
    fclose(fp);

    /* reopen the file */
    fp = fopen(FNAME, "rb");
    if (fp == NULL)
    {
        printf("Error opening file %s\n", FNAME);
        exit(1);
    }

    /* extract the word */
    word = getw(fp);
    if (ferror(fp))
        printf("Error reading file\n");
    else
        printf("Successful read: word = %d\n", word);

    /* clean up */
    fclose(fp);
    unlink(FNAME);
    return 0;
}
```

qsort

Funktion — Sortiert die Elemente eines Array in aufsteigender Reihenfolge.

Syntax

```
#include <stdlib.h>
void qsort(void *base, size_t nelem, size_t width,
           int (*fcmp)(const void *, const void *));
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *qsort* ist die Implementation einer Variante des QuickSort-Algorithmus zum Sortieren der Elemente eines Array.

Die Elemente können (wie bei *bsearch*) aus beliebigen Datenkonstrukten bestehen, weil die Funktion nur die Abfolge der Vergleiche festlegt. *qsort* erwartet die folgenden Argumente:

base einen Zeiger auf das erste Element (0) des zu sortierenden Array;
nelem die Anzahl der zu sortierenden Elemente;
width die Größe eines einzelnen Elements (in Byte);
fcmp einen Zeiger auf die Vergleichsfunktion.

Die Vergleichsfunktion muß vom Programmierer folgendermaßen definiert werden:

```
int <Name> (e_typ *elem1, e_typ *elem2);
```

Ihr werden bei jedem Aufruf durch *qsort* zwei Zeiger auf Elemente des zu sortierenden Array übergeben, sie muß die entsprechenden Elemente miteinander vergleichen und folgende Integer zurückliefern:

```
*elem1 <  *elem2   Rückgabewert < 0
*elem1 == *elem2   Rückgabewert = 0
*elem1 >  *elem2   Rückgabewert > 0
```

Für absteigende Reihenfolge braucht man diese Definition nur »umzudrehen« (das heißt für **elem1 < *elem2* einen Wert größer 0 und für **elem1 > *elem2* einen Wert kleiner Null zurückzuliefern).

Rückgabewert Keiner.

Siehe auch *bsearch, lsearch*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int sort_function( const void *a, const void *b);

char list[5][4] = { "cat", "car", "cab", "cap", "can"};

int main(void)
{
   int  x;

   qsort((void *)list, 5, sizeof(list[0]),
         sort_function);
   for (x = 0; x < 5; x++)
      printf("%s\n", list[x]);
   return 0;
}

int sort_function( const void *a, const void *b)
{
   return( strcmp((char *)a,(char *)b) );
}
```

raise

Funktion Sendet ein Software-Signal zum ausführenden Programm.

Syntax
```
#include <signal.h>
int raise(int sig);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung Mit der Funktion *raise* lassen sich Software-Signale vom Typ *sig* an das Programm senden. Wenn das Programm einen Signal-Handler für den durch *sig* angegebenen Signaltyp installiert hat, wird die Routine ausgeführt. Wenn kein Handler installiert ist, wird die Standardaktion für den jeweiligen Signaltyp ausgeführt.

Die gegenwärtig in signal.h definierten Signaltypen und Standardaktionen sind:

Signal	Bedeutung	Standardaktion
SIGARBRT	Abnormal termination	_exit(3)
SIGFPE	Bad floating point operation	_exit(1)
SIGILL	Illegal instruction	_exit(1)
SIGINT	Control break interrupt	Aufruf des Interrupt 0x23
SIGSEGV	Invalid access to storage	_exit(1)
SIGTERM	Request for program termination	_exit(1)

Die Signaltypen SIGABRT und SIGTERM werden nicht direkt von C++ oder DOS erzeugt, sie können jedoch mit *raise* erzeugt werden.

SIGILL und SIGSEGV können von den Prozessoren 8086 und 8088 nicht asynchron erzeugt werden, wohl aber von einigen anderen Prozessoren. Weitere Einzelheiten zu Software-Signalen finden Sie in diesem Handbuch unter *signal*.

Rückgabewert *raise* liefert bei fehlerfreier Ausführung den Wert 0 zurück. Für undefinierte Werte von *sig* ist der Rückgabewert ungleich Null.

Siehe auch *abort*, *signal*

Beispiel
```
#include <signal.h>

int main(void)
{
   int a, b;

   a = 10;
   b = 0;
   if (b == 0)
```

```
   /* preempt divide by zero error */
      raise(SIGFPE);
   a = a / b;
   return 0;
}
```

rand

Funktion: Erzeugt eine »Zufallszahl«.

Syntax:

```
#include <stdlib.h>
int rand(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung: *rand* verwendet ein multiplikatives Kongruenzverfahren (Multiplikation mit anschließender Modulo-Operation), um »Zufallszahlen« im Bereich von 0 bis RAND_MAX zu erzeugen. Die symbolische Konstante RAND_MAX ist in stdlib.h definiert; ihr Wert ist 2^{15}-1 Die Periodendauer des Generators beträgt 2^{32}.

Rückgabewert: *rand* liefert die erzeugte Zufallszahl zurück.

Siehe auch: *random, randomize, srand*

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   int i;

   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
      printf("%d\n", rand() % 100);
   return 0;
}
```

randbrd

Funktion: Liest Records einer Random-Datei.

Syntax:

```
#include <dos.h>
int randbrd(struct fcb *fcb, int rcnt);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *randbrd* verwendet die DOS-Funktion 0x27, um die durch *rcnt* angegebene Anzahl von Records einer Datei in den Speicher zu lesen. Die Datei wird über einen »file control block« (FCB) bezeichnet, auf den *fcb* zeigt, und muß zum Zeitpunkt des Aufrufs von *randbrd* bereits offen sein; gelesene Blöcke werden ab der Adresse gespeichert, die durch die aktuelle *disk transfer address* (DTA) gesetzt ist.

Die tatsächliche Zahl gelesener Records kann dem Random-Record-Feld des FCB entnommen werden. Dieses Feld wird um die Zahl der tatsächlich gelesenen Records erhöht.

Rückgabewert *randbrd* liefert Werte im Bereich von 0 bis 3 zurück:

0 Die durch *rcnt* angegebene Anzahl von Records konnte fehlerfrei gelesen werden.

1 Dateiende erreicht, der letzte Record wurde vollständig gelesen.

2 Das Lesen von Records würde über eine Segmentgrenze im Speicher hinausgehen (es werden so viele Records wie möglich bearbeitet, danach folgt der Abbruch).

3 Dateiende erreicht, der letzte gelesene Record ist nicht vollständig.

Siehe auch *getdta, randbwr, setdta*

Beispiel

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   char far *save_dta;
   char line[80], buffer[256];
   struct fcb blk;
   int i, result;

   /* get user input file name for dta */
   printf("Enter drive and file name "
          "(no path - i.e. a:file.dat)\n");
   gets(line);

   /* put file name in fcb */
   if (!parsfnm(line, &blk, 1))
   {
      printf("Error in call to parsfnm\n");
      exit(1);
   }
   printf("Drive #%d  File: %s\n\n", blk.fcb_drive,
          blk.fcb_name);
```

```
   /* open file with DOS FCB open file */
   bdosptr(0x0F, &blk, 0);

   /* save old dta, and set new one */
   save_dta = getdta();
   setdta(buffer);

   /* set up info for the new dta */
   blk.fcb_recsize = 128;
   blk.fcb_random = 0L;
   result = randbrd(&blk, 1);

   /* check results from randbrd */
   if (!result)
      printf("Read OK\n\n");
   else
   {
      perror("Error during read");
      exit(1);
   }

   /* read in data from the new dta */
   printf("The first 128 characters are:\n");
   for (i=0; i<128; i++)
      putchar(buffer[i]);

   /* restore previous dta */
   setdta(save_dta);

   return 0;
}
```

randbwr

Funktion: Schreibt Records in eine Random-Datei.

Syntax:

```
#include <dos.h>
int randbwr(struct fcb *fcb, int rcnt);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung: *randbwr* verwendet die DOS-Funktion 0x28, um die durch *rcnt* angegebene Anzahl von Records in eine Datei zu schreiben. Die Datei wird über einen »file control block« (FCB) bezeichnet, auf den *fcb* zeigt, und muß zum Zeitpunkt des Aufrufs von *randbwr* bereits offen sein.

Durch die Angabe von 0 für *rcnt* wird die Datei auf die im Random-Record-Feld des FCB angegebene Länge verkürzt.

Die tatsächliche Zahl geschriebener Records kann dem Random-Record-Feld des FCB entnommen werden. Dieses Feld

wird um die Zahl der tatsächlich geschriebenen Records erhöht.

Rückgabewert *randbwr* liefert Werte im Bereich von 0 bis 2 zurück:

0 Die durch *rcnt* angegebene Anzahl von Records konnte fehlerfrei geschrieben werden.

1 Nicht genügend Platz auf der Diskette (diese Prüfung erfolgt *vor* dem Schreiben des ersten Records – wenn nicht genügend Platz vorhanden ist, bricht *randbwr* sofort ab).

2 Das Schreiben von Records würde über eine Segmentgrenze im Speicher hinausgehen (es werden so viele Records wie möglich bearbeitet, danach folgt der Abbruch).

Siehe auch *randbrd*

Beispiel

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   char far *save_dta;
   char line[80];
   char buffer[256] = "RANDBWR test!";
   struct fcb blk;
   int result;

   /* get new file name from user */
   printf("Enter a file name to create "
          "(no path - ie. a:file.dat\n");
   gets(line);

   /* parse the new file name to the dta */
   parsfnm(line,&blk,1);
   printf("Drive #%d  File: %s\n", blk.fcb_drive,
          blk.fcb_name);

   /* request DOS services to create file */
   if (bdosptr(0x16, &blk, 0) == -1)
   {
      perror("Error creating file");
      exit(1);
   }

   /* save old dta and set new dta */
   save_dta = getdta();
   setdta(buffer);

   /* write new records */
   blk.fcb_recsize = 256;
```

```
   blk.fcb_random = 0L;
   result = randbwr(&blk, 1);

   if (!result)
      printf("Write OK\n");
   else
   {
      perror("Disk error");
      exit(1);
   }

   /* request DOS services to close the file */
   if (bdosptr(0x10, &blk, 0) == -1)
   {
      perror("Error closing file");
      exit(1);
   }

   /* reset the old dta */
   setdta(save_dta);

   return 0;
}
```

random

Funktion — Erzeugt eine »Zufallszahl« innerhalb eines bestimmten Bereichs.

Syntax

```
#include <stdlib.h>
int random(int num);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *random* liefert eine »Zufallszahl« im Bereich von 0 bis *num*-1 zurück und ist als Makro in stdlib.h definiert. Sowohl *num* als auch die zurückgelieferte Zufallszahl sind Integer.

Rückgabewert — *random* liefert einen **int**-Wert im Bereich von 0 bis *num*-1 zurück.

Siehe auch — *rand, randomize, srand*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

/* prints a random number in the range 0 to 99 */
int main(void)
{
```

```
   randomize();
   printf("Random number in the 0-99 range: %d\n",
          random (100));
   return 0;
}
```

randomize

Funktion: Initialisiert den »Zufallszahlen-Generator«.

Syntax:

```
#include <stdlib.h>
#include <time.h>
void randomize(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung: *randomize* initialisiert den »Zufallszahlen-Generator« und ist als Makro definiert.

Da es die Funktion *time* benutzt, sollte time.h in Programme mit aufgenommen werden, die *randomize* verwenden.

Rückgabewert: Keiner.

Siehe auch: *rand, random, srand*

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(void)
{
   int i;

   randomize();
   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
       printf("%d\n", rand() % 100);
   return 0;
}
```

_read, _dos_read

Funktion: Liest Daten aus einer Datei.

Syntax:

```
#include <io.h>
int _read(int handle, void *buf, unsigned len);

#include <dos.h>
unsigned _dos_read(int handle, void far *buf,
                   unsigned *nread);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_read liest *len* Bytes aus der durch *handle* bezeichneten Datei in den durch *buf* angegebenen Speicherbereich. Ist die Datei im Modus *text* eröffnet, dann entfernt *_read* keine CR-Zeichen.

handle ist das über einen Aufruf von *creat*, *open*, *dup* oder *dup2* zugeordnete Datei-Handle.

_dos_read verwendet die DOS-Funktion 0x3F, um *len* Bytes aus der durch *handle* bezeichneten Datei in den durch den **far**-Zeiger *buf* angegebenen Speicherbereich zu lesen. Die Anzahl der gelesenen Bytes wird an der Adresse gespeichert, auf die *nread* zeigt. Im Falle eines Fehlers oder beim Erreichen des Dateiendes kann diese Zahl kleiner als *len* sein.

Die Funktion *_dos_read* entfernt keine Wagenrücklaufzeichen (CR), da sie ausschließlich mit binären Dateien arbeitet.

handle ist bei *_dos_read* das über einen Aufruf von *_dos_creat*, *_dos_creatnew*, oder *_dos_open* zugeordnete Datei-Handle.

Bei Diskettendateien beginnt der Lesevorgang ab der aktuellen Position innerhalb der Datei. Nach Beendigung des Lesevorgangs wird der Dateizeiger um die Anzahl der gelesenen Bytes erhöht. Bei Gerätedateien ist eine »Position« nicht definiert – hier lesen *_read* und *_dos_read* direkt vom entsprechenden Gerät.

_read und *_dos_read* können maximal 65534 Bytes lesen, da 65535 (0xFFFF) dem Wert -1 und damit dem Fehlerindikator entspricht.

Rückgabewert

Bei fehlerfreier Ausführung hat *_dos_read* den Rückgabewert 0. Andernfalls liefert die Funktion den DOS-Fehlercode zurück und setzt die globale Variable *errno*.

_read liefert bei fehlerfreier Ausführung die Anzahl der gelesenen Bytes als (positiven) Integer zurück. Wenn das Dateiende erreicht bzw. überschritten wurde, hat *_read* den Rückgabewert 0.

Im Falle eines Fehlers hat *_read* den Rückgabewert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

EACCES	Zugriff nicht gestattet
EBADF	Ungültiges Handle

Siehe auch

_open, *read*, *_write*

Beispiel 1

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
```

```
#include <process.h>
#include <sys\stat.h>

int main(void)
{
   void *buf;
   int handle, bytes;

   buf = malloc(10);

/*
   Looks for a file in the current directory
   named TEST.$$$ and attempts to read 10
   bytes from it. To use this example you
   should create the file TEST.$$$
*/
   if ((handle =
       open("TEST.$$$", O_RDONLY | O_BINARY,
            S_IWRITE | S_IREAD)) == -1)
   {
      printf("Error Opening File\n");
      exit(1);
   }

   if ((bytes = _read(handle, buf, 10)) == -1) {
      printf("Read Failed.\n");
      exit(1);
   }
   else {
      printf("_read: %d bytes read.\n", bytes);
   }
   return 0;
}
```

Beispiel 2

```
#include <stdio.h>
#include <fcntl.h>
#include <dos.h>

int main(void)
{
  int handle;
  unsigned bytes;
  char buf[10];

  /* Looks for a file in the current directory named
     TEST.$$$ and attempts to read 10 bytes from it.
     To use this example you should create the file
     TEST.$$$
  */
  if (_dos_open("TEST.$$$", O_RDONLY, &handle) != 0) {
    perror("Unable to open TEST.$$$");
    return 1;
```

```
  }
  if (_dos_read(handle, buf, 10, &bytes) != 0) {
    perror("Unable to read from TEST.$$$");
    return 1;
  }
  else
    printf("_dos_read: %d bytes read.\n", bytes);
  return 0;
}
```

read

Funktion

Liest Daten aus einer Datei.

Syntax

```
#include <io.h>
int read(int handle, void *buf, unsigned len);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

read liest *len* Bytes aus der durch *handle* bezeichneten Datei in den durch *buf* angegebenen Speicherbereich. Wird die Datei im Modus *text* eröffnet, dann entfernt *read* Wagenrücklaufzeichen und interpretiert das Zeichen Strg-Z als »Dateiende«.

handle ist das über einen Aufruf von *creat*, *open*, *dup* oder *dup2* zugeordnete Datei-Handle.

Bei Diskettendateien beginnt der Lesevorgang ab der aktuellen Position innerhalb der Datei. Nach Beendigung des Lesevorgangs wird der Dateizeiger um die Anzahl der gelesenen Bytes erhöht. Bei Gerätedateien ist eine »Position« nicht definiert – hier liest *read* direkt vom entsprechenden Gerät.

read kann maximal 65534 Bytes lesen, da 65535 (0xFFFF) dem Wert -1 und damit dem Fehlerindikator entspricht .

Rückgabewert

Bei fehlerfreier Ausführung liefert *read* die Anzahl der gelesenen Bytes als positiven Integer zurück. Wagenrückläufe und Strg-Z werden dabei nicht mitgezählt. Wenn das Dateiende erreicht bzw. überschritten wurde, ist der Rückgabewert 0.

Im Fall eines Fehlers ist der Rückgabewert -1 und die globale Variable *errno* wird auf einen der folgenden Werte gesetzt:

EACCES	Zugriff nicht gestattet
EBADF	Ungültiges Handle

Siehe auch

open, *_read*, *write*

Beispiel

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
```

```
#include <process.h>
#include <sys\stat.h>

int main(void)
{
   void *buf;
   int handle, bytes;

   buf = malloc(10);

/*
   Looks for a file in the current directory named
   TEST.$$$ and attempts to read 10 bytes from it.
   To use this example you should create the file
   TEST.$$$
*/
   if ((handle =
      open("TEST.$$$", O_RDONLY | O_BINARY,
           S_IWRITE | S_IREAD)) == -1)
   {
      printf("Error Opening File\n");
      exit(1);
   }

   if ((bytes = read(handle, buf, 10)) == -1) {
      printf("Read Failed.\n");
      exit(1);
   }
   else {
      printf("Read: %d bytes read.\n", bytes);
   }
   return 0;
}
```

readdir

Funktion — Liest den aktuellen Eintrag aus einem Verzeichnis-Stream.

Syntax

```
#include <dirent.h>
struct dirent readdir(DIR *dirp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *readdir* ist unter UNIX auf POSIX-Systemen verfügbar. Die Funktion liest den aktuellen Verzeichniseintrag aus dem durch *dirp* bezeichneten Verzeichnis-Stream. Der Verzeichnis-Stream wird auf den nächsten Eintrag gesetzt.

readdir liefert einen Zeiger auf eine Struktur des Typs *dirent* zurück. Bei jedem neuen Aufruf von *readdir* für den gleichen Verzeichnis-Stream wird diese Struktur überschrieben.

Die in dirent.h definiert Struktur des Typs *dirent* entspricht einem einzelnen Verzeichniseintrag. Sie enthält mehrere Elemente, von denen der Programmierer aber nur auf eines zugreifen kann:

```
char d_name[];
```

Das **char** -Array *d_name* enthält den nullterminierten Dateinamen für den aktuellen Verzeichniseintrag. Die Größe dieses Array ist unbestimmt. Sie können die Länge des Dateinamens mit der Funktion *strlen* ermitteln.

Zurückgeliefert werden alle Verzeichniseinträge einschließlich Unterverzeichnissen, ».«- und »..«-Einträgen, Systemdateien, verborgenen Dateien und Datenträgernamen. Unbenutzte oder gelöschte Verzeichniseinträge werden nicht zurückgeliefert.

Verzeichniseinträge können zwar während des Lesens eines Verzeichnis-Streams erstellt oder gelöscht werden, es ist jedoch nicht sicher, ob *readdir* die betreffenden Einträge zurückliefert. Um sicherzustellen, daß *readdir* den aktuellen Status des Verzeichnisses zurückliefert, muß der Verzeichnis-Stream mit *rewinddir* auf den ersten Eintrag zurückgesetzt oder das Verzeichnis mit *opendir* neu geöffnet werden.

Rückgabewert Bei fehlerfreier Ausführung liefert *readdir* einen Zeiger auf den aktuellen Verzeichniseintrag zurück. *readdir* hat den Rückgabewert NULL, wenn das Verzeichnisende erreicht wird oder *dirp* keinen offenen Verzeichnis-Stream bezeichnet.

Siehe auch *closedir, opendir, rewinddir*

Beispiel Siehe das Beispiel zu *opendir*.

real

Funktion Liefert den Realteil einer komplexen Zahl oder konvertiert eine BCD-Zahl in einen Wert vom Typ **float**, **double** oder **long double**.

Syntax

komplexe Version:
```
#include <complex.h>
double real(complex x);
```

bcd-Version:
```
#include <bcd.h>
double real(bcd x);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		x

Beschreibung Die zu komplexen Zahlen gehörigen Daten bestehen aus zwei Gleitkommazahlen. *real* liefert diejenige Zahl, die als Realteil betrachtet wird.

Mit *real* kann auch eine BCD-Zahl in einen Wert vom Typ **float**, **double** oder **long double** zurückkonvertiert werden.

Rückgabewert *real* liefert den Realteil der komplexen Zahl zurück.

Siehe auch *bcd, complex, imag*

Beispiel
```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary real part = "
        << imag(z) << "\n";
   cout << "z has complex conjugate = "
        << conj(z) << "\n";
   return 0;
}
```

realloc

Funktion Ändert die Größe eines belegten Blocks im Hauptspeicher.

Syntax
```
#include <stdlib.h>
void *realloc(void *block, size_t size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *realloc* verändert die Größe eines zuvor belegten Speicherblocks auf *size* Bytes. *block* muß auf die Startadresse eines Speicherblocks zeigen, der zuvor mit einem Aufruf von *malloc*, *calloc* oder *realloc* ermittelt wurde, *size* gibt die neue Größe des Blocks an. Wenn *block* ein NULL-Zeiger ist, verhält sich *realloc* wie *malloc*. Falls notwendig, wird der gesamte Block im Speicher verschoben.

Rückgabewert *realloc* gibt die Adresse des betroffenen Speicherblocks zurück, die von der bisherigen Adresse abweichen kann.

Ist die Neubelegung nicht möglich oder size==0, gibt *realloc* NULL zurück.

Siehe auch *calloc, farrealloc, free, malloc*

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>

int main(void)
{
   char *str;

   /* allocate memory for string */
   str = (char *) malloc(10);

   /* copy "Hello" into string */
   strcpy(str, "Hello");

   printf("String is %s\n  Address is %p\n",str, str);
          str = (char *) realloc(str, 20);
   printf("String is %s\n  New address is %p\n", str,
          str);

   /* free memory */
   free(str);

   return 0;
}
```

rectangle

Funktion Zeichnet ein Rechteck.

Syntax

```
#include <graphics.h>
void far rectangle(int left, int top,
                   int right, int bottom);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *rectangle* zeichnet ein Rechteck, dessen obere linke Ecke durch (*left, top*) und dessen untere rechte Ecke durch (*right, bottom*) festgelegt ist, wobei die durch *setlinestyle* gesetzten Parameter (Linienart, -dicke und -farbe) verwendet werden.

Rückgabewert Keiner.

Siehe auch *bar, bar3d, setcolor, setlinestyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int left, top, right, bottom;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   left = getmaxx() / 2 - 50;
   top = getmaxy() / 2 - 50;
   right = getmaxx() / 2 + 50;
   bottom = getmaxy() / 2 + 50;

   /* draw a rectangle */
   rectangle(left,top,right,bottom);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

registerbgidriver

Funktion: Registriert einen eingebundenen oder geladenen Grafiktreiber.

Syntax:

```
#include <graphics.h>
int registerbgidriver(void (*driver)(void));
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung: *registerbgidriver* ermöglicht es einem Benutzer, eine Treiberdatei zu laden und den Treiber zu »registrieren«. Sobald dessen Speicheradresse an *registerbgidriver* übergeben wurde, verwendet *initgraph* den somit registrierten Treiber. Ein benutzerregi-

strierter Treiber kann von der Diskette auf den Heap geladen oder mit BINOBJ.EXE in eine .OBJ-Datei konvertiert und in die .EXE-Datei gelinkt werden.

Der Aufruf von *registerbgidriver* informiert das Grafiksystem darüber, daß der Treiber, auf den *driver* zeigt, zur Linkzeit eingebunden wurde. Diese Routine überprüft den hinzugelinkten Code des angegegebenen Treibers; ist er gültig, so wird er in internen Tabellen registriert.

Durch die Verwendung des Namens eines hinzugelinkten Treibers in einem Aufruf von *registerbgidriver* weisen Sie außerdem den Compiler (und Linker) an, die .OBJ-Datei mit diesem öffentlichen Namen hinzuzulinken.

Rückgabewert

Nach einer fehlerfreien Ausführung liefert *registerbgidriver* die Kennziffer des »registrierten« Grafiktreibers zurück. Negative Funktionsergebnisse entsprechen den Statuscodes des Grafikpakets und zeigen einen Fehler an.

Wenn ein vom Programmierer erstellter Treiber registriert wird, *muß* das Ergebnis von *registerbgidriver* als Treibernummer an *initgraph* übergeben werden.

Siehe auch

graphresult, initgraph, installuserdriver, registerbgifont

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* register a driver that was added
      into graphics.lib
   */
   errorcode = registerbgidriver(EGAVGA_driver);

   /* report any registration errors */
   if (errorcode < 0)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
```

```
  /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

registerbgifont

Funktion: Registiert einen als .OBJ-Datei eingebundenen Vektor-Fonts.

Syntax:

```
#include <graphics.h>
int registerbgifont(void (*font)(void));
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung: *registerbgifont* erwartet die Adresse einer als .OBJ-Datei in das Programm eingebundenen Vektorschrift und trägt diesen Font nach einer Prüfung in eine interne Tabelle für *settextstyle* ein. Auf diese Weise »registrierte« Fonts werden nicht mehr als .CHR-Dateien auf der Diskette gesucht und auf den Heap geladen, sondern direkt verwendet.

Da Fonts nur indirekt von anderen Routinen angesprochen werden, erfüllt der Aufruf der »Registrierfunktion« einen weiteren Zweck: er teilt dem Linker von Borland C++ mit, daß die entsprechende .OBJ-Datei mit in das Programm aufgenommen werden muß.

Wenn ein vom Programmierer erstellter Font registriert wird, *muß* das Ergebnis von *registerbgifont* als Fontnummer an *settextstyle* übergeben werden.

Rückgabewert: Nach einer fehlerfreien Ausführung liefert *registerbgifont* die Nummer des »registrierten« Grafik-Fonts zurück. Negative Funktionsergebnisse entsprechen den Statuscodes des Grafikpakets und zeigen einen Fehler an.

Siehe auch *graphresult, initgraph, installuserdriver, registerbgidriver, settextstyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* register a font file that
      was added into graphics.lib
   */
   errorcode = registerbgifont(triplex_font);

   /* report any registration errors */
   if (errorcode < 0)
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* select the registered font */
   settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
```

```
   /* output some text */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, "The TRIPLEX FONT");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

remove

Funktion Löscht eine Datei.

Syntax
```
#include <stdio.h>
int remove(const char *filename);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *remove* löscht die durch *filename* angegebene Datei, wobei *filename* (wie beim DOS-Befehl DELETE) einen vollständigen Suchweg und/oder einen Laufwerksbezeichner enthalten kann – die Jokerzeichen * und ? sind dagegen nicht erlaubt.

Wenn die Datei offen ist, müssen Sie sie erst schließen, bevor Sie sie mit *remove* löschen können.

remove ist in *stdio.h* als Makro in der Form *unlink*(**filename*) definiert.

Rückgabewert Bei fehlerfreier Ausführung liefert *remove* den Wert 0 zurück. Im Falle eines Fehlers ist der Rückgabewert -1 und die globale Variable *errno* bekommt einen der folgenden Werte:

ENOENT	Suchweg/Datei nicht gefunden
EACCES	Zugriff nicht gestattet

Siehe auch *unlink*

Beispiel
```
#include <stdio.h>

int main(void)
{
   char file[80];

   /* prompt for file name to delete */
   printf("File to delete: ");
   gets(file);

   /* delete the file */
   if (remove(file) == 0)
```

```
      printf("Removed %s.\n",file);
   else
      perror("remove");

   return 0;
}
```

rename

Funktion Ändert den Namen einer Datei.

Syntax
```
#include <stdio.h>
int rename(const char *oldname, const char *newname);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *rename* ändert den Namen der Datei *oldname* in *newname*. Beide Namen können vollständige Suchwege und/oder Laufwerksbezeichner enthalten. Die Angabe unterschiedlicher Laufwerke in *oldname* und *newname* ist nicht zulässig.

Es ist mit *rename* möglich, Dateien von einem Verzeichnis in ein anderes zu verlagern. Die Angabe von Jokerzeichen ist nicht zulässig.

Rückgabewert Bei fehlerfreier Ausführung liefert *rename* den Wert 0 zurück; ansonsten ist das Funktionsergebnis -1 und *errno* bekommt einen der folgenden Werte:

ENOENT	Suchweg/Datei nicht gefunden
EACCES	Zugriff nicht gestattet
ENOTSAM	Ziel- und Quell-Laufwerk sind nicht identisch

Beispiel
```
#include <stdio.h>

int main(void)
{
   char oldname[80], newname[80];

   /* prompt for file to rename and new name */
   printf("File to rename: ");
   gets(oldname);
   printf("New name: ");
   gets(newname);

   /* Rename the file */
   if (rename(oldname, newname) == 0)
      printf("Renamed %   s to % s.\n", oldname, newname);
```

```
   else
      perror("rename");

   return 0;
}
```

restorecrtmode

Funktion

Setzt wieder den Videomodus, der vor dem Aufruf von *initgraph* aktiv war.

Syntax

```
#include <graphics.h>
void far restorecrtmode(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

initgraph hält beim Start des Grafikpakets den ursprünglich aktiven Videomodus fest; mit *restorecrtmode* und *setgraphmode* können Sie deshalb zwischen Text- und Grafikmodus wechseln. *textmode* sollte für diesen Zweck nicht verwendet werden, diese Funktion dient nur zum Umschalten zwischen verschiedenen Textmodi.

Vor dem Aufruf von *restorecrtmode* muß das Grafikpaket mit *initgraph* aktiviert worden sein – ansonsten ist der »vorherige« Videomodus undefiniert.

Rückgabewert

Keiner.

Siehe auch

getgraphmode, initgraph, setgraphmode

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
```

```
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "Press any key to exit graphics:");
   getch();

   /* restore system to text mode */
   restorecrtmode();
   printf("We're now in text mode.\n");
   printf("Press any key to return to "
          "graphics mode:");
   getch();

   /* return to graphics mode */
   setgraphmode(getgraphmode());

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "We're back in graphics mode.");
   outtextxy(x, y+textheight("W"),
             "Press any key to halt:");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

rewind

Funktion

Setzt die Position eines Dateizeigers auf den Anfang des Stream.

Syntax

```
#include <stdio.h>
void rewind(FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *rewind* ist äquivalent zu *fseek*(*stream*, 0L, SEEK_SET). Anders als *fseek* löscht *rewind* nicht nur das Dateiende-Flag, sondern auch eventuell gesetzte Fehler-Flags.

In Dateien, die für Lese- *und* Schreiboperationen eröffnet worden sind, kann nach *rewind* zwischen »Lesen« und »Schreiben« gewechselt werden.

Rückgabewert Keiner.

Siehe auch *fopen*, *fseek*, *ftell*

Beispiel Siehe auch das Beispiel zu *fseek*.

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
   FILE *fp;
   char *fname = "TXXXXXX", *newname, first;

   newname = mktemp(fname);
   fp = fopen(newname,"w+");
   fprintf(fp,"abcdefghijklmnopqrstuvwxyz");
   rewind(fp);
   fscanf(fp,"%c",&first);
   printf("The first character is: %c\n",first);
   fclose(fp);
   remove(newname);

   return 0;
}
```

rewinddir

Funktion Setzt den Verzeichnis-Stream auf den ersten Eintrag zurück.

Syntax

```
#include <dirent.h>
void rewinddir(DIR *dirp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *rewinddir* ist unter UNIX auf POSIX-Systemen verfügbar. Die Funktion setzt den Verzeichnis-Stream *dirp* auf den ersten Eintrag im Verzeichnis zurück. *rewinddir* stellt sicher, daß der Verzeichnis-Stream den aktuellen Status der Einträge widerspiegelt: Es werden alle Einträge berücksichtigt, die seit dem letzten Aufruf von *opendir* oder *rewinddir* für diesen Verzeichnis-Stream erzeugt oder gelöscht wurden.

Rückgabewert Keiner.

Siehe auch *closedir*, *opendir*, *readdir*

Beispiel Siehe das Beispiel zu *opendir*.

rmdir

Funktion Löscht ein DOS-Verzeichnis.

Syntax
```
#include <dir.h>
int rmdir(const char *path);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *rmdir* löscht das in *path* angegebene Verzeichnis und kann nur angewendet werden, wenn dieses Verzeichnis

- keine Einträge enthält,
- nicht das aktuelle Arbeitsverzeichnis ist,
- nicht das Stammverzeichnis der Diskette/Festplatte darstellt.

Rückgabewert *rmdir* liefert 0 zurück, wenn die Operation fehlerfrei ausgeführt werden konnte. Bei Fehlern ist der Rückgabewert -1 und *errno* bekommt einen der folgenden Werte:

EACCES	Zugriff nicht gestattet
ENOENT	Suchweg nicht gefunden

Siehe auch *chdir*, *getcurdir*, *getcwd*, *mkdir*

Beispiel
```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>

#define DIRNAME "testdir.$$$"

int main(void)
{
   int stat;

   stat = mkdir(DIRNAME);
   if (!stat)
          printf("Directory created\n");
   else
   {
      printf("Unable to create directory\n");
```

```
        exit(1);
    }

    getch();
    system("dir/p");
    getch();

    stat = rmdir(DIRNAME);
    if (!stat)
          printf("\nDirectory deleted\n");
    else
    {
          perror("\nUnable to delete directory\n");
        exit(1);
    }

    return 0;
}
```

rmtmp

Funktion: Löscht temporäre Dateien.

Syntax:

```
#include <stdio.h>
int rmtmp(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung: *rmtmp* schließt und löscht alle offenen, temporären Datei-Streams, die vorher mit *tmpfile* erzeugt wurden. Das aktuelle Verzeichnis muß dasselbe sein, in dem die Dateien erzeugt wurden – andernfalls werden die Dateien nicht gelöscht.

Rückgabewert: *rmtmp* liefert die Anzahl der geschlossenen und gelöschten Dateien zurück.

Siehe auch: *tmpfile*

Beispiel:

```
#include <stdio.h>
#include <process.h>

void main()
{
  FILE *stream;
  int i;

  /* Create temporary files */
  for (i = 1; i <= 10; i++)
```

```
  {
    if ((stream = tmpfile()) == NULL)
      perror("Could not open temporary file\n");
    else
      printf("Temporary file %d created\n", i);
  }
  /* Remove temporary files */
  if (stream != NULL)
    printf("%d temporary files deleted\n", rmtmp());
}
```

_rotl

Funktion

Rotiert einen vorzeichenlosen Integerwert bitweise nach links.

Syntax

```
#include <stdlib.h>
unsigned _rotl(unsigned value, int count);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_rotl rotiert den als *value* übergebenen vorzeichenlosen Integerwert um *count* Bitpositionen nach links.

Rückgabewert

Die Funktion liefert das Ergebnis der Rotation als **unsigned int** zurück.

Siehe auch

_lrotl, _lrotr, _rotr

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

/* rotl example */

int rotl_example(void)
{
   unsigned value, result;

   value = 32767;
   result = _rotl(value, 1);
   printf("The value %u rotated left one bit is: "
          "%u\n", value, result);
   return 0;
}

/* rotr example */

int rotr_example(void)
{
   unsigned value, result;

   value = 32767;
```

```
   result = _rotr(value, 1);
   printf("The value %u rotated right one bit is: "
          "%u\n", value, result);
   return 0;
}

int main(void)
{
   rotl_example();
   rotr_example();
   return 0;
}
```

_rotr

Funktion — Rotiert einen vorzeichenlosen Integerwert bitweise nach rechts.

Syntax

```
#include <stdlib.h>
unsigned _rotr(unsigned value, int count);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *_rotr* rotiert den als *value* übergebenen vorzeichenlosen Integerwert um *count* Bitpositionen nach rechts.

Rückgabewert — *_rotr* liefert das Ergebnis der Rotation als **unsigned int** zurück.

Siehe auch — *_lrotl, _lrotr, _rotl*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

/* rotl example */

int rotl_example(void)
{
   unsigned value, result;

   value = 32767;
   result = _rotl(value, 1);
   printf("The value %u rotated left one bit is: "
          "%u\n", value, result);
   return 0;
}

/* rotr example */

int rotr_example(void)
```

```
{
   unsigned value, result;

   value = 32767;
   result = _rotr(value, 1);
   printf("The value %u rotated right one bit is: "
          "%u\n", value, result);
   return 0;
}

int main(void)
{
   rotl_example();
   rotr_example();
   return 0;
}
```

sbrk

Funktion Verändert die Größe des Datensegments.

Syntax
```
#include <alloc.h>
void *sbrk(int incr);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x			

Beschreibung *sbrk* fügt *incr* Bytes zum *Break*-Wert des Programms hinzu und ändert die belegte Speichergröße entsprechend. Wenn *incr* ein negativer Wert ist, wird der belegte Speicher verkleinert.

sbrk liefert einen Fehler zurück und ändert nichts an der Speicherbelegung, wenn durch die Änderung mehr Speicher belegt würde, als zulässig ist.

Rückgabewert Im Fehlerfall wird der Wert -1 zurückgeliefert, das Datensegment bleibt unverändert und die globale Variable *errno* bekommt den Wert

ENOMEM Nicht genug Platz im Hauptspeicher

Die fehlerfreie Ausführung von *sbrk* liefert den vorherigen *Break*-Wert zurück.

Siehe auch *brk*

Beispiel
```
#include <stdio.h>
#include <alloc.h>

int main(void)
{
   printf("Changing allocation with sbrk()\n");
```

```
    printf("Before sbrk() call: %lu bytes free\n",
           (unsigned long) coreleft());
    sbrk(1000);
    printf(" After sbrk() call: %lu bytes free\n",
            (unsigned long) coreleft());
    return 0;
}
```

scanf

Funktion Liest und formatiert Eingabe aus dem Stream *stdin*.

Syntax

```
#include <stdio.h>
int scanf(const char *format [, adress, ...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung *scanf* liest eine Reihe von Eingabefeldern zeichenweise aus dem Stream *stdin*. Auf jedes Feld wird dann die entsprechende Formatangabe angewendet, die *scanf* über den Zeiger *format* als Formatstring übergeben wurde. Der Formatstring legt die Anzahl der Eingaben und ihr Format fest. Gelesene Eingaben werden an der Adresse, die in dem *format* nachfolgenden Parameter übergeben wurde, gespeichert. Nach der Analyse der ersten Formatanweisung wird ein Eingabefeld gelesen, konvertiert und an der entsprechenden Adresse gespeichert.

Die Anzahl von Eingabefeldern muß der Anzahl von Formatangaben und Adressen genau entsprechen.

Der Formatstring

Der Formatstring ist in jeder Variante von *scanf* (*cscanf*, *fscanf*, *sscanf*, *vscanf*, *vfscanf*, *vsscanf*) obligatorisch – über ihn wird die Art der Konvertierung, die Formatierung und die Anzahl der Argumente festgelegt. Argumente, die durch den Formatstring nicht erfaßt (d.h. überzählig) sind, werden ignoriert – wenn *scanf* aufgrund des Formatstrings dagegen mehr Argumente erwartet als angegeben sind, ergeben sich merkwürdige bis katastrophale Resultate.

Hinweis: scanf führt oft zu überraschenden Ergebnissen, wenn die Eingabe vom erwarteten Muster abweicht. Sie müssen genau angeben, wie *scanf* das Zeilenende behandeln soll. Die Kombination von *gets* oder *fgets* mit einem folgenden *sscanf* ist sicher und einfach und deshalb vorzuziehen.

Der Formatstring ist ein nullterminierter String, der drei Arten von Informationen enthält:

- »Whitespace-Zeichen« (Leerzeichen, Tabulatoren (\t) und Zeilenvorschübe (\n)) weisen *scanf* an, dieses und alle folgen-

den Whitespace-Zeichen zwar zu lesen, aber nicht zu speichern;

- »normale Zeichen«, d.h. die anderen ASCII-Zeichen mit Ausnahme des Prozentzeichens (%), die im Prinzip dieselbe Wirkung wie »Whitespace-Zeichen« haben. Die Angabe eines Kommas weist *scanf* beispielsweise an, zwischen zwei Angaben eine beliebige Anzahl von Kommas (mindestens aber eines) zu lesen, aber nicht zu speichern.

- Formatierungsanweisungen, die *scanf* mitteilen, welche Art von Eingabe erwartet wird, wie sie konvertiert und wo sie anschließend gespeichert werden soll.

Nachfolgende Whitespace-Zeichen (einschließlich Zeilenvorschüben) werden nur gelesen, wenn dies im Formatstring angegeben wird.

Formatierungsanweisungen
Formatierungsanweisungen lassen sich durch das folgende Syntaxdiagramm darstellen:

% [*] [Breite] [F|N] [h|l|L] Typ

Jede Anweisung beginnt mit einem Prozentzeichen (%). Auf dieses Zeichen folgen (in der angegebenen Reihenfolge):

Anweisung	Wirkung oder Bedeutung
[*]	Unterdrückt eine Zuweisung über den korrespondierenden Adreßparameter (optional).
[Breite]	Eine (optionale) Angabe über die maximal zu lesende Zeichenzahl. Wenn vor Erreichen dieser Zeichenzahl ein »Whitespace-Zeichen« oder ein anderes nicht auswertbares Zeichen gelesen wird, beendet *scanf* die Eingabe dieses Wertes von selbst.
[F|N]	Eine (optionale) Angabe der Größe des Adreßparameters (N = **near**-Zeiger, *F* = **far**-Zeiger). Nur notwendig, wenn die Größe des Parameters von der »Standardgröße« abweicht (**far**-Zeiger in kleinen Speichermodellen usw.).
[h|L|l]	Eine (optionale) Angabe über die Größe der Variablen, auf die der korrespondierende Adreßparameter zeigt (h = **short int**, l = **long** oder **double**, L = **long double**).
Typ	Die Angabe des numerischen Typs (siehe nächste Tabelle). Diese Angabe ist obligatorisch.

Die folgende Aufstellung geht davon aus, daß außer dem Typ keine weiteren Angaben existieren (Kombinationen von Typ, dem Operator *, Breite usw. werden auf den nächsten Seiten beschrieben):

Typ	Eingabe	Adreßparameter
numerische Werte		
d	Integer (dezimal)	Zeiger auf **int**, (int *arg)
D	Integer (dezimal)	Zeiger auf **long**, (long *arg)
o	Integer (oktal)	Zeiger auf **int**, (int *arg)
O	Integer (oktal)	Zeiger auf **long**, (long *arg)
i	Integer (dezimal) Integer (oktal) Integer (hexadezimal)	Zeiger auf **int**, (int *arg)
I	Integer (dezimal) Integer (oktal)	
u	Integer (dezimal)	Zeiger auf **unsigned int**, (unsigned *arg)
U	Integer (dezimal)	Zeiger auf unsigned **long** (unsigned long *arg)
x	Integer (hexadezimal)	Zeiger auf **unsigned int**, (unsigned *arg)
X	Integer (hexadezimal)	Zeiger auf **unsigned long** (unsigned long *arg)
e	Gleitkomma	Zeiger auf **float** (float *arg)
E	Gleitkomma	Zeiger auf **float** (float *arg)
f	Gleitkomma	Zeiger auf **float** (float *arg)
g	Gleitkomma	Zeiger auf **float** (float *arg)
G	Gleitkomma	Zeiger auf **float** (float *arg)
Zeichen		
s	String	Zeiger auf einen String (char arg[]).
c	Zeichen	Zeiger auf ein einzelnes Zeichen (char *arg). Wenn hier eine *Breite* angegeben ist (z.B. %5c): Zeiger auf ein **char**-Array entsprechender Größe (char arg[Breite]).
%	Zeichen %	Keine Konvertierung, sondern Speicherung des Zeichens %.
Zeiger		
n		Zeiger auf **int** (int *arg) – speichert die Anzahl bis jetzt gelesener Zeichen an der Adresse, auf die der Zeiger zeigt.
p	YYYY oder YYYY:ZZZZ (hexadezimal)	Zeiger auf ein beliebiges Objekt (**far *** oder **near***) – die Eingabe wird in hexadezimaler Form interpretiert, die Standardvorgabe ist vom verwendeten Speichermodell abhängig.

Eingabefelder
scanf ordnet jeder Formatanweisung ein Eingabefeld zu, das zeichenweise analysiert wird. Auf den Abschluß der Analyse folgt die Speicherung des Ergebnisses über den korrespondierenden Adreßparameter. Ein Eingabefeld umfaßt:

- entweder alle Zeichen bis zum nächsten »Whitespace-Zeichen«, wobei das »Whitespace-Zeichen« nicht mit eingeschlossen ist;
- oder alle Zeichen bis zum ersten Zeichen, das sich über die aktuelle Formatanweisung nicht auswerten läßt (wie zum Beispiel die Zeichen »8« und »9« bei der Eingabe eines oktalen Wertes);
- oder maximal so viele Zeichen, wie in der aktuellen Formatanweisung als Breite angegeben ist.

Konventionen
Je nach Art der Formatanweisung gelten einige zusätzliche Konventionen.

%c-Format – die Anweisung *%c* liest das jeweils nächste Zeichen, wobei »Whitespace-Zeichen« nicht gesondert berücksichtigt (oder überlesen) werden.

Um Whitespace-Zeichen zu überspringen und das folgende Zeichen zu lesen, sollte man deshalb die Anweisung *%1s* verwenden.

%Bc-Format (B = Breite) – *scanf* erwartet einen Zeiger auf ein **char**-Array mit *B* Elementen als Adreßargumente (char *arg*[B]).

%s-Format – *scanf* erwartet einen Zeiger auf ein **char**-Array als Adreßargument (char *arg*[]). Das Array muß Platz für ein zusätzliches Nullzeichen bieten. Die Eingabe wird durch ein Leerzeichen oder einen Zeilenvorschub beendet. *scanf* speichert automatisch ein Nullzeichen als letztes Zeichen des Strings.

%[Suchmenge]-Format – *scanf* erwartet (wie bei %s) einen Zeiger auf ein **char**-Array als Adreßparameter. *Suchmenge* ist eine Folge von Zeichen (alle ASCII-Zeichen außer [und]), aus denen die Eingabe bestehen muß, und kann für invertierte Suche mit einem Caret (^) eingeleitet werden.

Um das Caret selbst in die Suchmenge einzubeziehen, darf es nicht an erster Stelle stehen.

Im folgenden einige Beispiele dazu:

%[abcd]	sucht im Eingabefeld nach den Zeichen *a*, *b*, *c* und *d*;
%[^abcd]	sucht im Eingabefeld nach allen Zeichen außer *a*, *b* , *c* oder *d*.

Das Eingabefeld wird durch »Whitespace-Zeichen« nicht beendet, sondern nur durch ein Zeichen, das nicht zur Suchmenge (bzw. zur invertierten Suchmenge) gehört. Anstelle einer direkten Aufzählung kann hier auch ein Bereich verwendet werden.

Die folgenden Angaben sind äquivalent:

```
%[0123456789]
%[0-9]
```

Mehrere Bereiche werden ohne spezielle Trennzeichen direkt aufeinanderfolgend angegeben. Im folgenden einige Beispiele dazu:

%[A-Z]	sämtliche Großbuchstaben
%[A-Za-z]	sämtliche Groß- und Kleinbuchstaben
%[A-Za-zäöüÄÖÜß]	sämtliche Buchstaben inklusive der Umlaute und des ß.

Die Regeln zur Angabe von Bereichen sind recht einfach:

- Das Zeichen vor dem Bindestrich (-) muß einen ASCII-Code haben, der mindestens um eins niedriger ist als der des auf den Bindestrich folgenden Zeichens.
- Der Bindestrich darf weder das erste noch das letzte Zeichen des Aufzählung sein (andernfalls wird er nicht als Bereichsangabe, sondern als Bindestrich gewertet).
- Das Zeichen vor und das Zeichen nach dem Bindestrich muß jeweils den Beginn bzw. das Ende eines Bereichs darstellen, dürfen also nicht gleichzeitig Beginn und Ende zweier verschiedener Bereiche sein.

In allen anderen Fällen wird der Bindestrich selbst als Zeichen (und nicht als Bereichsangabe) gewertet. Einige Beispiele dazu:

%[-+*/]	die vier arithmetischen Operatoren
%[z-a]	die Zeichen »z«, »-« und »a«
%[+0-9-A-F]	das Zeichen »+«, die Ziffern von 0 bis 9, das Zeichen »-« und die Buchstaben von A bis F
%[+0-9A-F-]	dieselben Zeichen wie im vorigen Beispiel
%[^-0-9+A-F]	sämtliche Zeichen außer »-«, den Ziffern 0 bis 9, dem Zeichen »+« und den Buchstaben A bis F.

%e, *%E* , *%f* , *%g* und *%G* (Gleitkomma) Konvertierungen – Gleitkommazahlen müssen dem folgenden Syntaxdiagramm entsprechen:

```
[+|-] ddddddd [.] dddd [E|e] [+/-] ddd
```

d steht hier für dezimale, oktale oder hexadezimale Ziffern, in eckige Klammern eingefaßte Teile sind optional.

Zusätzlich werden *+INF*, *-INF*, *+NAN* und *-NAN* als Gleitkommazahlen erkannt. Hierfür sind Vorzeichen und Großschreibung erforderlich.

%d, *%i*, *%o*, *%x*, *%D*, *%I*, *%O*, *%X*, *%c* und *%n* – in allen Fällen, in denen Zeiger auf **char** , **int** oder **long** erlaubt sind, sind auch Zeiger auf die jeweiligen **unsigned** -Typen erlaubt.

Unterdrückung einer Zuordnung mit *
Folgt der Operator * direkt auf das % einer Formatanweisung, wertet *scanf* die restlichen Elemente dieser Anweisung aus und analysiert die Eingabe, nimmt aber keine Zuweisung über das korrespondierende Adreßargument vor. (Hier wird zwarm dasselbe Zeichen wie für den Umleitungsoperator verwendet – ansonsten haben diese beiden Dinge aber nichts miteinander zu tun).

Die Unterdrückung von Zuweisungen bei *scanf* ist mit gewisser Vorsicht zu genießen – eine direkte Möglichkeit zur Prüfung, ob die »richtigen« Teile einer Eingabe dadurch übersprungen wurden, gibt es nicht.

Die Angabe der Breite
Die Breitenangabe erfolgt durch einen dezimalen Integer innerhalb der Formatanweisung, die zwischen dem % und dem Typ der zu erwartenden Eingabe steht (*%18s* legt beispielsweise fest, daß *scanf* einen String mit maximal 18 Zeichen einlesen soll).

Auswirkungen hat eine derartige Angabe nur dann, wenn das entsprechende Feld nicht aufgrund einer anderen Bedingung als beendet betrachtet wird (z.B. durch ein »Whitespace-Zeichen« oder ein nicht auswertbares Zeichen wie »8« oder »9« bei der Eingabe einer Oktalzahl).

F | *N* | *h* | *l* | *L* **– explizite Größenangaben**
Die Größenangaben *F* (**far**-Zeiger) und *N* (**near**-Zeiger) legen eine (vom verwendeten Speichermodell abweichende) Zeigergröße fest und werden nur für Adreßargumente verwendet.

Die Größenangaben *h*, *l* und *L* sind nur für Eingaben von Integern und Gleitkommawerten anwendbar – auf alle anderen Formate haben sie keine Wirkung.

Angabe	Wirkung
F	Das korrespondierende Adreßargument wird als **far**-Zeiger interpretiert.
N	Das korrespondierende Adreßargument wird als **near**-Zeiger interpretiert; kann im Speichermodell *huge* nicht verwendet werden.

Angabe	Wirkung
h	Für die Formate *d*, *i*, *o*, *u*, und *x*: Konvertierung der Eingabe in das Format **short int**. Für alle anderen Formate ohne Wirkung. Das Ergebnis wird in einem **short**-Objekt gespeichert.
l	Für die Formate *d*, *i*, *o*, *u*, und *x*: Konvertierung der Eingabe in das Format **long int** und Speicherung in einem **long**-Objekt; für die Formate *e*, *f* und *g*: Konvertierung der Eingabe in das Format **double** und Speicherung in einem **double**-Objekt. Für alle anderen Formate ohne Wirkung.
L	Für die Formate *e*, *f* und *g*: Konvertierung der Eingabe in das Format **long double** und Speicherung in einem entsprechenden Objekt. Für alle anderen Formate ohne Wirkung.

Abbruch einer Auswertung
scanf betrachtet ein Eingabefeld als beendet, wenn

- direkt auf das %-Zeichen einer Formatanweisung ein * folgt (Unterdrückung der Zuweisung) – hier wird das aktuelle Feld vollständig interpretiert, das Ergebnis aber nicht gespeichert;
- die durch *Breite* für dieses Feld angegebene Anzahl von Zeichen erreicht ist;
- das nächste Zeichen im aktuellen Format nicht auswertbar ist (wie beispielsweise ein »A« bei einem numerischen Format);
- das nächste Zeichen nicht in der Suchmenge bzw. in der invertierten Suchmenge aufgeführt ist.

In allen vier Fällen setzt *scanf* die Auswertung mit dem nächsten Eingabefeld fort (solange noch eine weitere Formatanweisung vorhanden ist); in allen Fällen außer dem ersten wird das Ergebnis der aktuellen Auswertung über das korrespondierende Adreßargument gespeichert.

Das Zeichen, das das aktuelle Feld beendet hat, wird als erstes Zeichen des nächsten Feldes (bzw. das erste Zeichen eines folgenden Lesebefehls) ausgewertet.

Die Funktion *scanf* endet, wenn eine der folgenden Bedingungen zutrifft:

- das nächste Zeichen des aktuellen Eingabefeldes ist EOF;
- es wurden so viele Eingabefelder gelesen und bearbeitet, wie durch den Formatstring angegeben;
- das nächste Zeichen eines Eingabefeldes stimmt nicht mit einem (als Trennzeichen vereinbarten) Zeichen des Formatstrings überein, beide Zeichen fallen nicht in die Klasse

»Whitespace-Zeichen«. Das ist beispielsweise der Fall, wenn *scanf* in der Form

```
scanf("% d-%d", &a, &b);
```

aufgerufen wird und der Benutzer die beiden numerischen Eingaben mit einem Komma voneinander trennt.

Zeichenfolgen innerhalb des Formatstrings, die nicht Teil einer Formatanweisung sind, müssen vom Benutzer in genau derselben Form eingegeben werden – ein Aufruf der Form

```
scanf("% dabc% d", &x, &u);
```

erwartet die Eingabe eines Integers, danach die Zeichenfolge *abc* und schließlich einen zweiten Integer. Jedes andere Zeichen außer *a* nach dem ersten Integer hat den Abbruch von *scanf* zur Folge, die »ungelesenen« Zeichen bleiben im Eingabepuffer des Systems und werden von folgenden Eingabebefehlen gelesen.

Rückgabewert

scanf liefert die Anzahl der fehlerfrei gelesenen Felder zurück, wobei mit * unterdrückte Felder nicht gezählt werden. Wenn überhaupt keine Zuweisungen erfolgt sind, ist der Rückgabewert 0.

Der Versuch, mit *scanf* über das Ende einer Datei bzw. über das Ende eines Strings hinauszulesen, ergibt den Wert EOF.

Siehe auch

atof, *cscanf*, *fscanf*, *getc*, *printf*, *sscanf*, *vfscanf*, *vscanf*, *vsscanf*

Beispiel

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
   char label[20];
   char name[20];
   int entries = 0;
   int loop, age;
   double salary;

   struct Entry_struct
   {
      char  name[20];
      int   age;
      float salary;
   } entry[20];

/* Input a label as a string of characters
   restricting to 20 characters
*/
   printf("\n\nPlease enter a label for the chart: ");
   scanf("% 20s", label);
```

```
   fflush(stdin);
     /* flush the input stream in case of bad input */

/* Input number of entries as an integer */
   printf("How many entries will there be? "
          "(less than 20) ");
   scanf("%d", &entries);
   fflush(stdin);
     /* flush the input stream in case of bad input */

/* input a name restricting input to only letters
   upper or lower case
*/
   for (loop=0;loop<entries;++loop)
   {
      printf("Entry %d\n", loop);
      printf("  Name   : ");
      scanf("%[A-Za-z]", entry[loop].name);
      fflush(stdin);
     /* flush the input stream in case of bad input */

/* input an age as an integer */
      printf("  Age    : ");
      scanf("%d", &entry[loop].age);
      fflush(stdin);
     /* flush the input stream in case of bad input */

/* input a salary as a float */
      printf("  Salary : ");
      scanf("%f", &entry[loop].salary);
      fflush(stdin);
     /* flush the input stream in case of bad input */
   }

/* Input a name, age and salary as a string,
   integer, and double
*/
   printf("\nPlease enter your name, "
          "age and salary\n");
   scanf("%20s %d %lf", name, &age, &salary);


/* Print out the data that was input */
   printf("\n\nTable %s\n",label);
   printf("Compiled by %s  age %d  $%15.2lf\n",
          name, age, salary);
   printf("----------------------------------------"
          "-----------\n");
```

```
   for (loop=0;loop<entries;++loop)
      printf("%4d | %-20s | %5d | %15.2lf\n",
         loop + 1,
         entry[loop].name,
         entry[loop].age,
         entry[loop].salary);
   printf("---------------------------------------"
          "----------\n");
   return 0;
}
```

_searchenv

Funktion: Sucht nach einer Datei im *Environment*-Verzeichnispfad.

Syntax:

```
#include <stdlib.h>
char *_searchenv(const char *file, const char *varname
                 char *buf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung: *_searchenv* sucht zuerst das aktuelle Verzeichnis nach einer Datei des angegebenen Namens (Parameter *file*) ab – wenn die Suche erfolglos bleibt, liest die Funktion die Pfadangaben in der *Environment*-Variable *varname* und durchsucht sämtliche darin enthaltenen Verzeichnisse der Reihe nach (typische *Environment*-Variablen sind PATH, LIB oder INCLUDE).

Wenn die Datei gefunden wurde, liefert *_searchenv* einen Zeiger auf den Puffer (*buf*) zurück, in dem der vollständige Dateiname (Laufwerk, Verzeichnis und Dateiname) gespeichert ist. Dieser String kann für einen folgenden Aufruf von *fopen*, oder *exec...* verwendet werden. Der Puffer muß groß genug sein, um jeden möglichen Dateinamen aufzunehmen.

Kann die Datei nicht gefunden werden, zeigt *buf* auf einen leeren String (der nur ein Null-Zeichen enthält).

Rückgabewert: Keiner.

Siehe auch: *exec..., _dos_findfirst, _dos_findnext, spawn..., system*

Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

char buf[_MAX_PATH];

int main(void)
{
  /* looks for TLINK */
  _searchenv("TLINK.EXE","PATH",buf);
```

```
  if (buf[0] == '\0')
    printf("TLINK.EXE not found\n");
  else
    printf("TLINK.EXE found in %s\n", buf);

  /* looks for non-existent file */
  _searchenv("NOTEXIST.FIL","PATH",buf);
  if (buf[0] == '\0')
    printf("NOTEXIST.FIL not found\n");
  else
    printf("NOTEXIST.FIL found in %s\n", buf);
  return 0;
}
```

Programmausgabe:

```
TLINK.EXE found in C:\BIN\TLINK.EXE
NOTEXIST.FIL not found
```

searchpath

Funktion: Sucht im DOS-Verzeichnispfad nach einer Datei.

Syntax:

```
#include <dir.h>
char *searchpath(const char *file);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung: *searchpath* sucht zuerst das aktuelle Verzeichnis nach einer Datei des angegebenen Namens (Parameter *file*) ab – wenn die Suche erfolglos bleibt, liest die Funktion den *Environment*-Eintrag PATH= und durchsucht sämtliche darüber angegebenen Verzeichnisse der Reihe nach.

Wenn die Datei gefunden wurde, liefert *searchpath* einen Zeiger auf den vollständigen Dateinamen (Laufwerk, Verzeichnis und Dateiname) zurück. Dieser String kann beispielsweise für einen folgenden Aufruf von *fopen* oder *exec...* verwendet werden.

Der zurückgelieferte Zeiger zeigt auf einen statischen Puffer, der bei jedem Aufruf von *searchpath* überschrieben wird.

Rückgabewert: Bei fehlerloser Ausführung wird ein Zeiger auf einen String mit dem gefundenen Dateinamen, im Fall eines Fehlers NULL zurückgeliefert.

Siehe auch: *exec..., findfirst, findnext, spawn..., system*

Beispiel:

```
#include <stdio.h>
#include <dir.h>

int main(void)
```

```
{
   char *p;

   /* Looks for TLINK and returns a pointer
      to the path  */
   p = searchpath("TLINK.EXE");
   printf("Search for TLINK.EXE : %s\n", p);

   /* Looks for non-existent file  */
   p = searchpath("NOTEXIST.FIL");
   printf("Search for NOTEXIST.FIL : %s\n", p);

   return 0;
}
```

Programm-ausgabe:

```
Search for TLINK.EXE : C:\BIN\TLINK.EXE
Search for NOTEXIST.FIL : (NULL)
```

sector

Funktion — Zeichnet ein ausgefülltes elliptisches »Kuchenstück«.

Syntax

```
#include <graphics.h>
void far sector(int x, int y, int stangle,
                int endangle, int xradius,
                int yradius);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *sector* zeichnet ein elliptisches »Kuchenstück«, dessen Mittelpunkt durch (*x*,*y*) angegeben ist. *xradius* und *yradius* legen den horizontalen bzw. vertikalen Radius fest. *stangle* legt den Startpunkt des mit *radius* gezogenen Ellipsenbogens fest, *endangle* sein Ende. Die Endpunkte des Ellipsenbogens werden mit dem Ellipsenmittelpunkt verbunden. Danach füllt *sector* das entstandene »Kuchenstück« mit dem durch *setfillpattern* bzw. *setfillstyle* festgelegten Muster und in der dort angegebenen Farbe.

Die Angabe beider Winkel erfolgt in Grad, wobei entgegen dem Uhrzeigersinn gezählt wird (0 Grad = horizontal rechts des Mittelpunktes, 90 Grad =senkrecht über dem Mittelpunkt, usw.)

Wenn während des Ausfüllens eines »Kuchenstücks« mit *sector* ein Fehler auftritt, liefert der nächste Aufruf von *graphresult* den Wert -6 (grNoScanMem).

Rückgabewert — Keiner.

Siehe auch — *arc*, *circle*, *ellipse*, *getarccoords*, *getaspectratio*, *graphresult*, *pieslice*, *setfillpattern*, *setfillstyle*, *setgraphbufsize*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;
   int stangle = 45, endangle = 135;
   int xrad = 100, yrad = 50;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i=EMPTY_FILL; i<USER_FILL; i++)
   {
      /* set the fill style */
      setfillstyle(i, getmaxcolor());

      /* draw the sector slice */
      sector(midx,midy, stangle,endangle, xrad,yrad);

      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

segread

Funktion — Liest die Segmentregister des Prozessors.

Syntax

```
#include <dos.h>
void segread(struct SREGS *segp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung — *segread* füllt eine Struktur des Typs SREGS mit den aktuellen Werten der Segmentregister und ist hauptsächlich für folgende Aufrufe von *intdosx* und *int86x* gedacht.

Rückgabewert — Keiner.

Siehe auch — *FP_OFF*, *int86*, *int86x*, *intdos*, *intdosx*, *MK_FP*, *movedata*

Beispiel

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   struct SREGS segs;

   segread(&segs);
   printf("Current segment register settings\n\n");
   printf("CS: %X   DS: %X\n", segs.cs, segs.ds);
   printf("ES: %X   SS: %X\n", segs.es, segs.ss);

   return 0;
}
```

setactivepage

Funktion — Legt fest, auf welcher Grafikspeicherseite folgende Zeichenbefehle arbeiten.

Syntax

```
#include <graphics.h>
void far setactivepage(int page);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *setactivepage* legt eine Grafikspeicherseite für nachfolgende Zeichenaktionen durch die Berechnung einer Basisadresse fest.

Mehrere Grafikspeicherseiten werden nur von EGA- und VGA-Karten sowie von der Hercules-Karte unterstützt. Mit Video-Adaptern dieser Typen ist es möglich, dem Benutzer eine Grafik-Seite zu zeigen, während »im Hintergrund« auf eine andere Seite gezeichnet wird.

Rückgabewert Keiner.

Siehe auch *setvisualpage*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
  /* select a driver and mode that supports
     multiple pages.
  */
  int gdriver = EGA, gmode = EGAHI, errorcode;
  int x, y, ht;

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)  /* an error occurred */
  {
    printf("Graphics error: %s\n",
           grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
  }

  x = getmaxx() / 2;
  y = getmaxy() / 2;
  ht = textheight("W");

  /*  select the off screen page for drawing */
  setactivepage(1);

  /* draw a line on page #1 */
  line(0, 0, getmaxx(), getmaxy());

  /* output a message on page #1 */
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(x, y, "This is page #1:");
  outtextxy(x, y+ht, "Press any key to halt:");

  /* select drawing to page #0 */
  setactivepage(0);

  /* output a message  on page #0 */
  outtextxy(x, y, "This is page #0.");
  outtextxy(x,y+ht, "Press any key to view page #1:");
  getch();
```

```
  /* select page #1 as the visible page */
  setvisualpage(1);

  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

setallpalette

Funktion — Setzt sämtliche Einträge einer Farbpalette.

Syntax

```
#include <graphics.h>
void far setallpalette(struct palettetype far
                       *palette);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *setallpalette* setzt mit der von *palette* übergebenen Struktur des Typs *palettetype* sämtliche Einträge einer Farbpalette, wobei *size* hier festlegt, wieviele Einträge (ab dem ersten, also der »Farbe 0«) gesetzt werden sollen.

Jedes Element des Array *colors* ersetzt einen Eintrag der Farbpalette und kann einen Wert von -1 bis 63 haben. Wie bei dem entsprechenden BIOS-Aufruf bedeutet der »Farbwert« -1, daß der korrespondierende Eintrag der Farbpalette unverändert bleiben soll.

Mit *setallpalette* können alle oder einzelne Farben der EGA/VGA-Palette verändert werden. Änderungen wirken sich sofort am Bildschirm aus: wird ein Paletteneintrag geändert, ändern alle Elemente mit diesem Farbwert ihre Farbe.

Die MAXCOLORS-Konstante und die Struktur des Typs *palettetype* sind in graphics.h folgendermaßen definiert:

```
#define MAXCOLORS 15

struct palettetype {
   unsigned char size;

   signed char colors[MAXCOLORS+1];
};
```

size enthält die Anzahl der tatsächlich benutzten Einträge des Array *colors*, sein Wert ist vom aktiven Grafikmodus und dem verwendeten Treiber abhängig (CGA: 4 Farben, EGA: 16 Farben usw). Die Elemente des Array *colors* (0 bis *size*-1) enthalten die in der aktiven Palette gespeicherten Bitmuster

(für CGA: Werte von 0 bis 3, für EGA und VGA: Werte von 0 bis 63).

Die folgenden symbolischen Konstanten sind in graphics.h für Farb-Bitmuster definiert:

CGA-Modus		EGA/VGA		
Name	Nr.	Name	Nr.	Farbe
BLACK	0	EGA_BLACK	0	schwarz
BLUE	1	EGA_BLUE	1	blau
GREEN	2	EGA_GREEN	2	grün
CYAN	3	EGA_CYAN	3	türkis
RED	4	EGA_RED	4	rot
MAGENTA	5	EGA_MAGENTA	5	violett
BROWN	6	EGA_LIGHTGRAY	7	hellgrau
LIGHTGRAY	7	EGA_BROWN	20	braun
DARKGRAY	8	EGA_DARKGRAY	56	dunkelgrau
LIGHTBLUE	9	EGA_LIGHTBLUE	57	hellblau
LIGHTGREEN	10	EGA_LIGHTGREEN	58	hellgrün
LIGHTCYAN	11	EGA_LIGHTCYAN	59	helles Türkis
LIGHTRED	12	EGA_LIGHTRED	60	hellrot
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61	helles Violett
YELLOW	14	EGA_YELLOW	62	gelb
WHITE	15	EGA_WHITE	63	weiß

Änderungen einer Palettenfarbe wirken sich sofort am Bildschirm aus und betreffen jedes Vorkommen dieser Farbe.

Hinweis: setallallpalette kann nicht zusammen mit dem IBM-8514-Treiber verwendet werden.

Rückgabewert Wenn *setallpalette* mit ungültigen Argumenten aufgerufen wird, bleibt die aktuelle Farbpalette unverändert; ein folgender Aufruf von *graphresult* liefert in diesem Fall den Wert -11 (grError).

Siehe auch *getpalette, getpalettesize, graphresult, setbkcolor, setcolor, setpalette*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   struct palettetype pal;
   int color, maxcolor, ht;
   int y = 10;
   char msg[80];
```

```
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   maxcolor = getmaxcolor();
   ht = 2 * textheight("W");

   /* grab a copy of the palette */
   getpalette(&pal);

   /* display the default palette colors */
   for (color=1; color<=maxcolor; color++)
   {
      setcolor(color);
      sprintf(msg, "Color: %d", color);
      outtextxy(1, y, msg);
      y += ht;
   }

   /* wait for a key */
   getch();

   /* black out the colors one by one */
   for (color=1; color<=maxcolor; color++)
   {
      setpalette(color, BLACK);
      getch();
   }

   /* restore the palette colors */
   setallpalette(&pal);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setaspectratio

Funktion — Verändert den Korrekturfaktor für das Höhen-/Seitenverhältnis des Bildschirms.

Syntax

```
#include <graphics.h>
void far setaspectratio(int xasp, int yasp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Jeder Grafiktreiber (.BGI) enthält einen Wert zur Korrektur des physikalischen Höhen-/Seitenverhältnisses des Bildschirms, der automatisch von *arc*, *circle* und *pieslice* verwendet wird, um auch optisch runde Kreise zu erhalten.

Mit *setaspectratio* lassen sich die von den Zeichenfunktionen verwendeten Korrekturfaktoren direkt setzen (z.B. für spezielle Video-Adapter); die Angabe eines neuen Faktors erfolgt analog zu den von *getaspectratio* gelieferten Ergebnissen (*yasp* =10000, *xasp* im entsprechenden Verhältnis dazu).

Rückgabewert — Keiner.

Siehe auch — *circle*, *getaspectratio*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xasp, yasp, midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
```

```
   setcolor(getmaxcolor());
   /* get current aspect ratio settings */
   getaspectratio(&xasp, &yasp);

   /* draw normal circle */
   circle(midx, midy, 100);
   getch();

   /* clear the screen */
   cleardevice();

   /* adjust the aspect for a wide circle */
   setaspectratio(xasp/2, yasp);
   circle(midx, midy, 100);
   getch();

   /* adjust the aspect for a narrow circle */
   cleardevice();
   setaspectratio(xasp, yasp/2);
   circle(midx, midy, 100);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setbkcolor

Funktion — Setzt die Hintergrundfarbe des Bildschirms.

Syntax

```
#include <graphics.h>
void far setbkcolor(int color);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *setbkcolor* setzt den Inhalt des ersten Eintrags der aktuellen Farbpalette auf den Wert von *color*. Für den Parameter *color* kann eine Nummer oder ein Name verwendet werden; hierfür sind die folgenden Konstanten in graphics.h als **enum** definiert:

Nr.	Name	Farbe	Nr.	Name	Farbe
0	BLACK	schwarz	8	DARKGRAY	dunkelgrau
1	BLUE	blau	9	LIGHTBLUE	hellblau
2	GREEN	grün	10	LIGHTGREEN	hellgrün
3	CYAN	türkis	11	LIGHTCYAN	helles Türkis
4	RED	rot	12	LIGHTRED	hellrot

Nr.	Name	Farbe	Nr.	Name	Farbe
5	MAGENTA	violett	13	LIGHTMAGENTA	helles Violett
6	BROWN	braun	14	YELLOW	gelb
7	LIGHTGRAY	hellgrau	15	WHITE	weiß

Folgende Aufrufe setzen Blau als Hintergrundfarbe:

```
setbkcolor (BLUE) /* or */ setbkcolor (1)
```

Auf CGA- und EGA-Karten ändert *setbkcolor* die Hintergrundfarbe durch Änderung des ersten Eintrags in der Palette.

Hinweis: Wenn bei einem EGA/VGA-System die Paletteneinträge mit *setpalette* oder *setallpalette* geändert werden, kann die Verwendung der symbolischen Namen zu falschen Farben führen. Der Grund hierfür ist, daß *setbkcolor* sich immer auf die jeweilige Nummer in der Palette bezieht und nicht auf eine bestimmte Farbe – außer der übergebene Parameter ist 0, wodurch die Hintergrundfarbe stets auf schwarz gesetzt wird.

Rückgabewert Keiner.

Siehe auch *getbkcolor, setallpalette, setcolor, setpalette*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* select a driver and mode that supports */
   /* multiple background colors.           */
   int gdriver = EGA, gmode = EGAHI, errorcode;
   int bkcol, maxcolor, x, y;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }
```

```
   /* maximum color index supported */
   maxcolor = getmaxcolor();

   /* for centering text messages */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* loop through the available colors */
   for (bkcol=0; bkcol<=maxcolor; bkcol++)
   {
      /* clear the screen */
      cleardevice();

      /* select a new background color */
      setbkcolor(bkcol);

      /* output a messsage */
      if (bkcol == WHITE)
         setcolor(EGA_BLUE);
      sprintf(msg, "Background color: %d", bkcol);
      outtextxy(x, y, msg);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

setblock, _dos_setblock

Funktion: Verändert die Größe eines zuvor belegten Speicherblocks.

Syntax:

```
#include <dos.h>
int setblock(unsigned segx, unsigned newsize);
unsigned _dos_setblock(unsigned newsize, unsigned segx,
                       unsigned *maxp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung: *setblock* und *_dos_setblock* verändern die Größe eines zuvor mit *allocmem* bzw. *_dos_allocmem* belegten Speichersegments. Die Startadresse des Blocks wird über den Parameter *segx* angegeben, *newsize* gibt die neue Größe in Paragraphen an.

Kann der Speicherblock nicht auf die angegebene Größe geändert werden, speichert *_dos_setblock* den Umfang des

größtmöglichen Bereichs an der durch *maxp* angegebenen Adresse.

Rückgabewert

Wenn ein Speicherbereich der gewünschten Größe belegt werden konnte, liefert *setblock* den Wert -1. Wenn nicht, wird die Größe des größten noch verfügbaren Speicherblocks (in Paragraphen) zurückgeliefert und die globale Variable *_doserrno* gesetzt.

_dos_setblock hat bei fehlerfreier Ausführung den Rückgabewert 0. Im Falle eines Fehlers wird der DOS-Fehlercode zurückgeliefert und die globale Variable *errno* wird auf folgenden Wert gesetzt:

ENOMEM	Nicht genug Platz im Hauptspeicher oder falsche Segementadresse

Siehe auch

allocmem, freemem

Beispiel 1

```
#include <dos.h>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   unsigned int size, segp;
   int stat;

   size = 64; /* (64 x 16) = 1024 bytes */
   stat = allocmem(size, &segp);
   if (stat == -1)
      printf("Allocated memory at segment: %X\n",
             segp);
   else
   {
      printf("Failed: maximum number of paragraphs "
             "available is %d\n", stat);
      exit(1);
   }

   stat = setblock(segp, size * 2);
   if (stat == -1)
      printf("Expanded memory block at segment: %X\n",
             segp);
   else
      printf("Failed: maximum number of paragraphs "
             "available is %d\n", stat);

   freemem(segp);

   return 0;
}
```

Beispiel 2

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
  unsigned int size, segp, err, maxb;
  size = 64; /* (64 x 16) = 1024 bytes */
  err = _dos_allocmem(size, &segp);
  if (err == 0)
    printf("Allocated memory at segment: %x\n", segp);
  else {
    perror("Unable to allocate block");
    printf("Maximum no. of paragraphs available is "
           "%u\n", segp);
    return 1;
  }
  if (_dos_setblock(size * 2, segp, &maxb) == 0)
    printf("Expanded memory block at segment: %X\n",
           segp);
  else {
    perror("Unable to expand block");
    printf("Maximum no. of paragraphs available is "
           "%u\n", maxb);
  }
  _dos_freemem(segp);
  return 0;
}
```

setbuf

Funktion

Explizite Zuordnung eines Puffers zu einem Stream.

Syntax

```
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung

setbuf ordnet der durch *stream* angegebenen Datei den Puffer *buf* anstelle des automatisch vergebenen Puffers zu. Die Minimalgröße von *buf* ist durch die in stdio.h definierte Konstante BUFSIZ festgelegt. Die Funktion kann nur angewendet werden, wenn die Datei bereits offen ist.

Die Angabe des Zeigerwertes NULL für *buf* bewirkt, daß folgende Ein-/Ausgaben über *stream* ungepuffert arbeiten, d.h. Ausgaben geschehen sofort, Eingaben werden direkt vom entsprechenden Gerät gelesen.

stdin und *stdout* arbeiten ungepuffert, wenn sie nicht umge-

leitet sind. Umleitungen bewirken dagegen normalerweise eine Pufferung, die mit *setbuf* verändert werden kann.

Die »Pufferung« von Ein- und Ausgaben bedeutet: Ausgaben werden zeichenweise zwischengespeichert und blockweise geschrieben; Eingaben werden blockweise von der Datei bzw. dem Gerät gelesen und danach zeichenweise aus dem Puffer geliefert.

setbuf darf nur direkt nach der Eröffnung (oder direkt nach einem *fseek*-Aufruf) auf einen Stream angewendet werden – ansonsten sind die Ergebnisse unvorhersehbar. Die Anwendung dieser Funktion auf ungepufferte Dateien ist dagegen problemlos und jederzeit möglich.

Hinweis: Für *buf* kann eine lokale Variable verwendet werden – allerdings sollte man in diesem Fall streng darauf achten, die Datei vor dem Ende der entsprechenden Funktion wieder zu schließen.

Rückgabewert Keiner.

Siehe auch *fflush*, *fopen*, *fseek*, *setvbuf*

Beispiel

```
#include <stdio.h>

/* BUFSIZ is defined in stdio.h */
char outbuf[BUFSIZ];

int main(void)
{
   /* attach a buffer to the standard output stream */
   setbuf(stdout, outbuf);

   /* put some characters into the buffer */
   puts("This is a test of buffered output.\n\n");
   puts("This output will go into outbuf\n");
   puts("and won't appear until the buffer\n");
   puts("fills up or we flush the stream.\n");

   /* flush the output buffer */
   fflush(stdout);

   return 0;
}
```

setcbrk

Funktion Legt fest, bei welchen Operationen DOS auf Strg Untbr prüft.

Syntax

```
#include <dos.h>
int setcbrk(int cbrkvalue);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *setcbrk* setzt mit der DOS-Funktion 0x33 die Art der Überprüfung von DOS auf Strg Untbr :

0 Überprüfung *aus* – DOS prüft nur bei Ein- und Ausgaben über Tastatur, Bildschirm, serielle Schnittstellen und Drucker auf einen Abbruch mit Strg Untbr .

1 Überprüfung *an* – DOS prüft bei jedem Systemaufruf, ob Strg Untbr eingegeben wurde.

Rückgabewert *setcbrk* liefert den als *cbrkvalue* übergebenen Wert zurück.

Siehe auch *getcbrk*

Beispiel
```
#include <dos.h>
#include <conio.h>
#include <stdio.h>

int main(void)
{
   int break_flag;

   printf("Enter 0 to turn control break off\n");
   printf("Enter 1 to turn control break on\n");

   break_flag = getch() - 0;

   setcbrk(break_flag);

   if (getcbrk())
      printf("Cntrl-brk flag is on\n");
   else
      printf("Cntrl-brk flag is off\n");
   return 0;
}
```

setcolor

Funktion Setzt die Farbe für folgende Zeichenaktionen.

Syntax
```
#include <graphics.h>
void far setcolor(int color);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *setcolor* legt fest, welcher Paletteneintrag zukünftig als Zeichenfarbe verwendet werden soll. *color* kann dabei Werte von 0 bis *getmaxcolor* haben.

Die Zeichenfarbe spezifiziert den Wert, auf den die Pixel beim Zeichen von Linien etc. gesetzt werden. Die folgende Tabelle zeigt die für CGA bzw. EGA definierten Konstanten für die Zeichenfarbe (Nr. bezeichnet die Nummer der Palette).

Konstante, die der Farbnummer zugewiesen wird (Pixelwert)			
Nr.	1	2	3
0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
2	CGA_GREEN	CGA_RED	CGA_BROWN
3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGREY

Nr.	Name	Farbe	Nr.	Name	Farbe
0	BLACK	schwarz	8	DARKGRAY	dunkelgrau
1	BLUE	blau	9	LIGHTBLUE	hellblau
2	GREEN	grün	10	LIGHTGREEN	hellgrün
3	CYAN	türkis	11	LIGHTCYAN	helles Türkis
4	RED	rot	12	LIGHTRED	hellrot
5	MAGENTA	violett	13	LIGHTMAGENTA	helles Violett
6	BROWN	braun	14	YELLOW	gelb
7	LIGHTGRAY	hellgrau	15	WHITE	weiß

Zur Auswahl der Zeichenfarbe wird entweder die Farbnummer oder der entsprechende symbolische Name an *setcolor* übergeben.

Ein Beispiel: Im Modus CGAC0 enthält die Farbpalette vier Einträge – die Hintergrundfarbe BLACK(0) sowie LIGHTGREEN(1), LIGHTRED(2) und YELLOW(3). Durch den Aufruf *setcolor*(3) bzw. *setcolor*(*CGA_YELLOW*) wird als Zeichenfarbe YELLOW festgelegt.

Rückgabewert Keiner

Siehe auch *getcolor*, *getmaxcolor*, *graphresult*, *setallpalette*, *setbkcolor*, *setpalette*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* select a driver and mode that supports */
   /* multiple drawing colors.               */
   int gdriver = EGA, gmode = EGAHI, errorcode;
   int color, maxcolor, x, y;
   char msg[80];
```

```
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* maximum color index supported */
   maxcolor = getmaxcolor();

   /* for centering text messages */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* loop through the available colors */
   for (color=1; color<=maxcolor; color++)
   {
      /* clear the screen */
      cleardevice();

      /* select a new background color */
      setcolor(color);

      /* output a messsage */
      sprintf(msg, "Color: %d", color);
      outtextxy(x, y, msg);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

_setcursortype

Funktion: Legt die Form des Cursors fest.

Syntax:

```
#include <conio.h>
void _setcursortype(int cur_t);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung Folgende Werte sind für *cur_t* möglich:

_NOCURSOR	Unsichtbar
_SOLIDCURSOR	Ausgefülltes Kästchen
_NORMALCURSOR	Unterstrich (Standard)

Rückgabewert Keiner.

Beispiel
```
#include <conio.h>
int main(void)
{
   /* Display the normal cursor */
   cprintf("\n\rNormal Cursor: "); getch();

   /* Turn off the cursor */
   _setcursortype(_NOCURSOR);
   cprintf("\n\rNo Cursor    : "); getch();

   /* Switch to a solid cursor */
   _setcursortype(_SOLIDCURSOR);
   cprintf("\n\rSolid Cursor : "); getch();

   /* Switch back to the normal cursor */
   _setcursortype(_NORMALCURSOR);
   cprintf("\n\rNormal Cursor: "); getch();

   return 0;
}
```

setdta

Funktion Setzt die *disk transfer address* (DTA) von DOS.

Syntax
```
#include <dos.h>
void setdta(char far *dta);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Die *disk transfer address* ist ein Bereich, der beispielsweise von *findfirst/findnext* und den »alten« DOS-Funktionen zur Übergabe von Daten verwendet wird.

setdta setzt die Startadresse der DTA auf die über *dta* angegebene Adresse.

Rückgabewert Keiner.

Siehe auch *getdta*

Beispiel
```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>
```

```
int main(void)
{
   char line[80], far *save_dta;
   char buffer[256] = "SETDTA test!";
   struct fcb blk;
   int result;

   /* get new file name from user */
   printf("Enter a file name to create:");
   gets(line);

   /* parse the new file name to the dta */
   parsfnm(line, &blk, 1);
   printf("%d %s\n", blk.fcb_drive, blk.fcb_name);

   /* request DOS services to create file */
   if (bdosptr(0x16, &blk, 0) == -1)
   {
      perror("Error creating file");
      exit(1);
   }

   /* save old dta and set new dta */
   save_dta = getdta();
   setdta(buffer);

   /* write new records */
   blk.fcb_recsize = 256;
   blk.fcb_random = 0L;
   result = randbwr(&blk, 1);
   printf("result = %d\n", result);

   if (!result)
      printf("Write OK\n");
   else
   {
      perror("Disk error");
      exit(1);
   }

   /* request DOS services to close the file */
   if (bdosptr(0x10, &blk, 0) == -1)
   {
      perror("Error closing file");
      exit(1);
   }

   /* reset the old dta */
   setdta(save_dta);
   return 0;
}
```

setfillpattern

Funktion
Ermöglicht die freie Definition von Mustern für Flächenfüllungen.

Syntax
```
#include <graphics.h>
void far setfillpattern(char far *upattern, int color);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung
setfillpattern legt das Muster und die Farbe für nachfolgende Aufrufe von *fillellipse, fillpoly, floodfill, bar, bar3d, sector* und *pieslice* fest. Die Funktion arbeitet wie *setfillstyle,* verwendet aber ein benutzerdefiniertes 8*8-Muster anstatt eines vordefinierten Musters.

upattern ist ein Zeiger auf eine Folge von 8 Bytes (normalerweise ein Array), bei dem jedes Bit für ein Pixel in der durch *color* festgelegten Farbe steht. Jedes Byte legt acht nebeneinanderliegende Punkte einer Bildschirmzeile fest, die Bytes selbst sind »untereinander« angeordnet. Wird ein Bit auf 1 gesetzt, wird das zugehörige Pixel mit der Farbe gefüllt.

Rückgabewert
Keiner.

Siehe auch
getfillpattern, getfillsettings, graphresult, sector, setfillstyle

Beispiel
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* a user defined fill pattern */
   char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x24,
                      0x24, 0x07, 0x00};

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
```

```
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();
   setcolor(getmaxcolor());

   /* select a user defined fill pattern */
   setfillpattern(pattern, getmaxcolor());

   /* fill the screen with the pattern */
   bar(0, 0, maxx, maxy);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setfillstyle

Funktion

Setzt ein vordefiniertes Muster und die Farbe zur Flächenfüllung.

Syntax

```
#include <graphics.h>
void far setfillstyle(int pattern, int color);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

setfillstyle setzt das aktuelle Muster und die dabei verwendete Farbe für Flächenfüllungen. Um ein durch den Programmierer definiertes Füllmuster (USER_FILL, Wert 12) zu setzen, muß *setfillpattern* verwendet werden.

Für das Füllmuster (*pattern*) definiert graphics.h mit dem Aufzählungstyp *fill_pattern* die folgenden Bitmuster:

Name	Wert	Beschreibung
EMPTY_FILL	0	Füllen mit der Hintergrundfarbe
SOLID_FILL	1	Füllen mit der Farbe *color*
LINE_FILL	2	----------
LTSLASH_FILL	3	/ / / /
SLASH_FILL	4	/ / / / mit dicken Linien
BKSLASH_FILL	5	\ \ \ \ mit dicken Linien

Name	Wert	Beschreibung
LTBKSLASH_FILL	6	\ \ \ \
HATCH_FILL	7	leicht schraffiert
XHATCH_FILL	8	stark schraffiert, überkreuzend
INTERLEAVE_FILL	9	gestrichelte Linien
WIDE_DOT_FILL	10	Punkte, weit auseinander
CLOSE_DOT_FILL	11	Punkte, dicht nebeneinander
USER_FILL	12	durch den Programmierer definiert

setfillstyle erwartet einen dieser Werte als *pattern* und setzt das entsprechende Bitmuster.

Alle Werte außer *EMPTY_FILL* verwenden die aktuelle Füllfarbe für die Flächenfüllung. *EMPTY_FILL* verwendet die aktuelle Hintergrundfarbe.

Ein Aufruf von *setfillstyle* mit einem ungültigen Wert für *pattern* wird als Fehler gewertet: das aktuelle Füllmuster bleibt in diesen Fällen unverändert, ein nachfolgender Aufruf von *graphresult* liefert den Wert -11 (grError).

Rückgabewert Keiner.

Siehe auch *bar*, *bar3d*, *fillpoly*, *floodfill*, *getfillsettings*, *graphresult*, *pieslice*, *sector*, *setfillpattern*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL",
                  "SOLID_FILL",
                  "LINE_FILL",
                  "LTSLASH_FILL",
                  "SLASH_FILL",
                  "BKSLASH_FILL",
                  "LTBKSLASH_FILL",
                  "HATCH_FILL",
                  "XHATCH_FILL",
                  "INTERLEAVE_FILL",
                  "WIDE_DOT_FILL",
                  "CLOSE_DOT_FILL",
                  "USER_FILL"
                };

int main(void)
{
```

```
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int style, midx, midy;
   char stylestr[40];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   for (style = EMPTY_FILL; style < USER_FILL;style++)
   {
      /* select the fill style */
      setfillstyle(style, getmaxcolor());

      /* convert style into a string */
      strcpy(stylestr, fname[style]);

      /* fill a bar */
      bar3d(0, 0, midx-10, midy, 0, 0);

      /* output a message */
      outtextxy(midx, midy, stylestr);

      /* wait for a key */
      getch();
      cleardevice();
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setgraphbufsize

Funktion — Legt die Größe des internen Grafikpuffers fest.

Syntax

```
#include <graphics.h>
unsigned far setgraphbufsize(unsigned bufsize);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — Manche Grafikroutinen, wie die zur Flächenfüllung und zum Zeichnen von Polygonen, verwenden einen internen Puffer, der von *initgraph* beim Start des Grafikpakets auf dem Heap belegt und von *closegraph* wieder freigegeben wird.

Standardmäßig umfaßt dieser Puffer 4096 Bytes – über einen Aufruf von *setgraphbufsize* vor dem Start des Grafikpakets läßt er sich beliebig verkleinern (auf minimal 0 Bytes) oder vergrößern. *setgraphbufsize* teilt *initgraph* mit, wieviel Speicher für diesen internen Buffer bereitzustellen ist, wenn es *_graphgetmem* aufruft. Eine Vergrößerung ist angebracht, wenn beispielsweise ein Aufruf von *floodfill* den Fehler -7 erzeugt: `Out of flood memory`.

setgraphbufsize muß vor *initgraph* aufgerufen werden. Nachdem *initgraph* aufgerufen wurde, werden alle weiteren *setgraphbufsize*-Aufrufe bis zum nächsten Aufruf von *closegraph* ignoriert.

Rückgabewert — *setgraphbufsize* liefert die vorherige Größe des Grafikpuffers zurück.

Siehe auch — *closegraph, _graphfreemem, _graphgetmem, initgraph, sector*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define BUFSIZE 1000 /* internal graphics
                        buffer size */

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y, oldsize;
   char msg[80];

   /* set the size of the internal graphics buffer */
   /* before making a call to initgraph.           */
   oldsize = setgraphbufsize(BUFSIZE);
```

```
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output some messages */
   sprintf(msg, "Graphics buffer size: %d", BUFSIZE);
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, msg);
   sprintf(msg, "Old graphics buffer size: %d",
           oldsize);
   outtextxy(x, y+textheight("W"), msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setgraphmode

Funktion

Schaltet in einen Grafikmodus um und löscht den Bildschirm.

Syntax

```
#include <graphics.h>
void far setgraphmode(int mode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

setgraphmode erwartet für *mode* einen für den benutzten Grafiktreiber gültigen Wert, löscht den Bildschirm, aktiviert den entsprechenden Grafikmodus und setzt alle Parameter des Grafikpakets (aktuelle Position, Palette, Farbe, usw.) auf ihre Standardvorgaben zurück. Diese Funktion kann zusammen mit der Funktion *restorecrtmode* zum Umschalten zwischen Text- und Grafikmodi benutzt werden.

Rückgabewert Keiner. Wenn *setgraphmode* mit einem Wert für *mode* aufgerufen wird, den der aktuelle Grafiktreiber nicht unterstützt, bleibt der Videomodus unverändert, ein folgender Aufruf von *graphresult* liefert -10 (*grInvalidMode*).

Siehe auch *getgraphmode*, *getmoderange*, *graphresult*, *initgraph*, *restorecrtmode*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "Press any key to exit graphics:");
   getch();

   /* restore system to text mode */
   restorecrtmode();
   printf("We're now in text mode.\n");
   printf("Press any key to return to graphics mode:");
   getch();

   /* return to graphics mode */
   setgraphmode(getgraphmode());
```

```
    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "We're back in graphics mode.");
    outtextxy(x, y+textheight("W"),
              "Press any key to halt:");

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

setjmp

Funktion Bereitet ein nicht-lokales *goto* vor.

Syntax
```
#include <setjmp.h>
int setjmp(jmp_buf jmpb);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *setjmp* speichert den Status der aktuellen Task in dem durch *jmpb* angegebenen Puffer und gibt 0 zurück. Ein Aufruf von *longjmp* zu einem späteren Zeitpunkt stellt diesen Zustand wieder her – die Task setzt die Arbeit an der Stelle fort, an der sich der Aufruf von *setjmp* befindet.

Ein Task-Status besteht aus folgenden Komponenten:

- Alle Segmentregister (CS, DS, ES, SS);
- Register-Variablen (SI, DI);
- Stackzeiger (SP);
- »Frame Pointer« für lokale Variablen (BP);
- Flags.

Ein Task-Status ist vollständig genug, um mit *setjmp* und *longjmp* Coroutinen zu implementieren.

setjmp muß vor *longjmp* aufgerufen werden. Die Routine, die *setjmp* aufgerufen und *jmpb* gesetzt hat, muß immer noch aktiv sein, bevor *longjmp* aufgerufen wird. Andernfalls sind die Ergebnisse unvorhersehbar.

setjmp ist besonders zur Behandlung von Fehlern und Ausnahmesituationen geeignet, die in maschinennahen Unterprogrammen auftreten.

Hinweis: Wenn Sie mit Overlays arbeiten, dürfen Sie *setjmp* und *longjmp* nicht verwenden, um Coroutinen zu implementieren. Zwar sichert *setjmp* alle von Coroutinen benötigten

Register und *longjmp* stellt sie wieder her, aber die Overlay-Verwaltung muß den Inhalt des Stack überwachen, und sie geht davon aus, daß es nur einen Stack gibt. Implementieren Sie aber Coroutinen, so gibt es zwei Stacks oder zwei Teile desselben Stack, und die Overlay-Verwaltung wird sie nicht korrekt behandeln.

Sie können durchaus mit Hintergrundroutinen arbeiten, diese dürfen aber keinerlei Overlay-Code benutzen. Sie dürfen auch nicht mit *setjmp* und *longjmp* von oder zu Hintergrundroutinen wechseln, wenn Sie mit Overlays arbeiten. Wenn Sie in Ihren Hintergrundroutinen keinen Overlay-Code verwenden, wird die Overlay-Verwaltung durch die Existenz eines Hintergrund-Stack aber nicht gestört.

Rückgabewert *setjmp* liefert beim Aufruf zum Setzen eines Sprungziels immer den Wert 0 zurück.

Der »Aufruf« von *setjmp* über *longjmp* setzt den Rückgabewert von *setjmp* auf einen Wert ungleich Null.

Siehe auch *longjmp*, *signal*

Beispiel

```
#include <stdio.h>
#include <process.h>
#include <setjmp.h>

void subroutine(void);

jmp_buf jumper;

int main(void)
{
   int value;

   value = setjmp(jumper);
   if (value != 0)
   {
      printf("Longjmp with value %d\n", value);
      exit(value);
   }
   printf("About to call subroutine ... \n");
   subroutine();
   return 0;
}

void subroutine(void)
{
   longjmp(jumper,1);
}
```

setlinestyle

Funktion Setzt die Linienart und -dicke für folgende Zeichenaktionen.

Syntax
```
#include <graphics.h>
void far setlinestyle(int linestyle,
                      unsigned upattern,
                      int thickness);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *setlinestyle* setzt die Linienart für folgende Zeichenoperationen mit *line*, *lineto*, *rectangle*, *drawpoly*, *arc*, *circle*, *ellipse*, *pieslice* usw.

Die Struktur vom Typ *linesettingstype* ist in graphics.h folgendermaßen definiert:

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
};
```

Für die Linienart (*linestyle*) definiert graphics.h mit dem Aufzählungstyp *line_styles* folgende Operatoren:

Name	Wert	Beschreibung
SOLID_LINE	0	durchgezogen
DOTTED_LINE	1	gepunktet
CENTER_LINE	2	Punkt Strich Punkt Strich...
DASHED_LINE	3	gestrichelt
USERBIT_LINE	4	benutzerdefiniert

Für die Dicke einer Linie (*thickness*) sind nur zwei Werte möglich, die ebenfalls in graphics.h definiert sind:

Name	Wert	Beschreibung
NORM_WIDTH	1	normale Dicke (1 Pixel)
THICK_WIDTH	3	»dicke« Dicke (3 Pixel)

Der Parameter *upattern* hat nur dann eine Wirkung, wenn *linestyle* den Wert USERBIT_LINE hat – in diesem Fall legen die einzelnen Bits das Punktmuster der Linie fest. Beispielsweise ergibt 0xFFFF für *upattern* eine durchgezogene Linie, 0x3333 oder 0x0F0F eine gestrichelte Linie usw.

Hinweis: Der Parameter *linestyle* wirkt sich nicht auf Kreisbögen, Kreise, Ellipsen, Kreis- oder Ellipsensektoren aus. Nur der Parameter *thickness* wird hierfür verwendet.

Wenn *setlinestyle* einen Wert von *upattern* für *linestyle* !=

USERBIT_LINE hat (d.h. ungleich 4 ist), wird der Parameter zwar ignoriert, muß aber trotzdem immer angegeben werden.

Rückgabewert Keiner. Aufrufe von *setlinestyle* mit einem Wert für *linestyle* außerhalb des gültigen Bereichs werden als Fehler gewertet: In diesen Fällen bleibt die aktuelle Linienart unverändert, ein nachfolgender Aufruf von *graphresult* liefert das Ergebnis -11.

Siehe auch *arc, bar3d, circle, drawpoly, ellipse, getlinesettings, graphresult, line, linerel, lineto, pieslice, rectangle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the line styles supported */
char *lname[] = {
   "SOLID_LINE",
   "DOTTED_LINE",
   "CENTER_LINE",
   "DASHED_LINE",
   "USERBIT_LINE"
   };

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   int style, midx, midy, userpat;
   char stylestr[40];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   /* a user defined line pattern */
   /* binary: "0000000000000001"  */
   userpat = 1;
```

```
   for (style=SOLID_LINE; style<=USERBIT_LINE;style++)
   {
      /* select the line style */
      setlinestyle(style, userpat, 1);

      /* convert style into a string */
      strcpy(stylestr, lname[style]);

      /* draw a line */
      line(0, 0, midx-10, midy);

      /* draw a rectangle */
      rectangle(0, 0, getmaxx(), getmaxy());

      /* output a message */
      outtextxy(midx, midy, stylestr);

      /* wait for a key */
      getch();
      cleardevice();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

setlocale

Funktion Auswahl einer locale-Variablen.

Syntax
```
#include <locale.h>
char *setlocale(int category, char *locale);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung Borlands C++ Compiler unterstützen momentan nur die locale-Variable "C". Der Aufruf von *setlocale* hat daher keine Auswirkungen. Folgende Werte sind für den Parameter *category* möglich:

LC_ALL
LC_COLLATE
LC_CTYPE
LC_MONETARY
LC_NUMERIC
LC_TIME

Rückgabewert Bei erfolgreicher Ausführung liefert *setlocale* einen String, der die locale-Variable bezeichnet, die vor dem Aufruf der Funktion aktiv war. Bei einem Mißerfolg wird ein NULL-Zeiger zurückgeliefert.

Siehe auch *localeconv*

Beispiel

```
#include <locale.h>
#include <stdio.h>

int main(void)
{
   char *old_locale;

   /* The only locale supported is "C" */
   old_locale = setlocale(LC_ALL,"C");
   printf("Old locale was %s\n",old_locale);

   return 0;
}
```

setmem

Funktion Füllt einen Speicherbereich mit einem Wert.

Syntax

```
#include <mem.h>
void setmem(void *dest, unsigned length, char value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung *setmem* setzt die ersten *length* Bytes von *dest* auf den Wert *value*.

Rückgabewert Keiner.

Siehe auch *memset, strset*

Beispiel

```
#include <stdio.h>
#include <alloc.h>
#include <mem.h>

int main(void)
{
   char *dest;

   dest = (char *) calloc(21, sizeof(char));
   setmem(dest, 20, 'c');
   printf("%s\n", dest);

   return 0;
}
```

setmode

Funktion Setzt den Modus einer offenen Datei.

Syntax
```
#include <fcntl.h>
int setmode(int handle, int amode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *setmode* setzt den Modus der durch *handle* bezeichneten Datei. Für *amode* kann genau eine der beiden in fcntl.h definierten Konstanten verwendet werden: O_BINARY oder O_TEXT.

Rückgabewert Bei fehlerfreier Ausführung liefert *setmode* den früheren Übersetzungsmodus zurück; ansonsten ist das Funktionsergebnis -1 und *errno* wird auf den folgenden Wert gesetzt:

EINVAL Ungültiges Argument

Siehe auch *_creat*, *creat*, *_open*, *open*

Beispiel
```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int result;

   result = setmode(fileno(stdprn), O_TEXT);
   if (result == -1)
      perror("Mode not available\n");
   else
      printf("Mode successfully switched\n");
   return 0;
}
```

set_new_handler

Funktion Bestimmt die Funktion, die aufgerufen wird, nachdem eine Speicherplatzanforderung nicht erfüllt werden konnte.

Syntax
```
#include <new.h>
void (*set_new_handler(void (*my_handler)() ))();
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		x

Beschreibung *set_new_handler* legt fest, welche Funktion aufgerufen wird, wenn *operator new* den angeforderten Speicherplatz nicht

reservieren kann. *operator new* liefert in diesem Fall standardmäßig Null zurück. Dieses Standardverhalten kann durch das Einrichten einer neuen Behandlungsroutine und den Aufruf von *set_new_handler* geändert werden.

Kann *new* den angeforderten Speicherplatz nicht reservieren, ruft es die vorher durch einen Aufruf von *set_new_handler* gesetzte Behandlungsroutine auf. *my_handler* legt die auszuführenden Aktionen fest. Nach der Rückkehr von *my_handler* versucht *new* erneut, den angeforderten Speicherplatz zu reservieren.

Im Idealfall gibt *my_handler* Speicherplatz frei, *new* stellt den Speicherplatz zur Verfügung und das Programm wird fortgesetzt. Kann der benötigte Speicherplatz nicht freigegeben werden, muß *my_handler* das Programm beenden (andernfalls würde eine Endlosschleife erzeugt).

Das Zurücksetzen auf die Standardbehandlungsroutine erfolgt mit *set_new_handler(0)*.

Der Einsatz von *set_new_handler* sollte nur Sonderfällen vorbehalten bleiben. In den meisten Fällen erreicht man diese Funktionalität besser durch Überladen von *operator new*().

Rückgabewert

set_new_handler liefert die vorherige Behandlungsroutine zurück, wenn eine definiert war. Standardmäßig ist keine Behandlungsroutine definiert.

Die benutzerdefinierte Funktion *my_handler* sollte keinen Wert zurückliefern.

Siehe auch

_new_handler (globale Variable)

Beispiel

```
#include <iostream.h>
#include <new.h>
#include <stdlib.h>

void mem_warn() {
  cerr << "\nCan't allocate!";
  exit(1);
  }

void main(void) {
  set_new_handler(mem_warn);

  char *ptr = new char[100];
  cout << "\nFirst allocation: ptr = " << hex
       << long(ptr);
  ptr = new char[64000U];
  cout << "\nFinal allocation: ptr = " << hex
       << long(ptr);
  set_new_handler(0);  // Reset to default.
}
```

Programmausgabe:

```
First allocation: ptr = 283e0f30
Cannot allocate!
```

setpalette

Funktion Ändert einen Eintrag der aktuellen Farbpalette.

Syntax
```
#include <graphics.h>
void far setpalette(int colornum, int color);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *setpalette* setzt einen einzelnen (durch *colornum* bezeichneten) Eintrag der Farbpalette auf die mit *color* bezeichnete Farbe. Beispielsweise ändert *setpalette* (*0,5*) den ersten Eintrag in der aktuellen Palette (die Hintergrundfarbe) auf die Farbe mit der Nummer 5. Gibt es in der aktuellen Palette *size* Einträge, so darf *colornum* im Bereich zwischen 0 und *size -1* liegen.

Mit *setpalette* können einzelne (oder alle) Farben der EGA/VGA-Palette verändert werden. Bei Verwendung einer CGA-Karte kann lediglich der erste Paletteneintrag (d.h. die Hintergrundfarbe) mit einem Aufruf von *setpalette* geändert werden.

Die folgenden symbolischen Farbkonstanten sind in der Header-Datei graphics.h definiert:

CGA-Modus		EGA/VGA		
Name	Wert	Name	Wert	Farbe
BLACK	0	EGA_BLACK	0	schwarz
BLUE	1	EGA_BLUE	1	blau
GREEN	2	EGA_GREEN	2	grün
CYAN	3	EGA_CYAN	3	türkis
RED	4	EGA_RED	4	rot
MAGENTA	5	EGA_MAGENTA	5	violett
BROWN	6	EGA_BROWN	20	braun
LIGHTGRAY	7	EGA_LIGHTGRAY	7	hellgrau
DARKGRAY	8	EGA_DARKGRAY	56	dunkelgrau
LIGHTBLUE	9	EGA_LIGHTBLUE	57	hellblau
LIGHTGREEN	10	EGA_LIGHTGREEN	58	hellgrün
LIGHTCYAN	11	EGA_LIGHTCYAN	59	helles Türkis
LIGHTRED	12	EGA_LIGHTRED	60	hellrot
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61	helles Violett
YELLOW	14	EGA_YELLOW	62	gelb
WHITE	15	EGA_WHITE	63	weiß

Die gültigen Farben sind abhängig vom aktuellen Grafiktreiber und Grafikmodus.

Änderungen wirken sich sofort am Bildschirm aus: wird ein Paletteneintrag geändert, ändern alle Elemente mit diesem Farbwert ihre Farbe.

Hinweis: *setpalette* kann nicht zusammen mit dem IBM-8514-Adapter verwendet werden. Verwenden Sie stattdessen *setrgbpalette*.

Rückgabewert Keiner. Wenn *setpalette* mit ungültigen Parametern aufgerufen wird, bleibt die aktuelle Farbpalette unverändert; ein folgender Aufruf von *graphresult* liefert in diesem Fall den Wert -11.

Siehe auch *getpalette, graphresult, setallpalette, setbkcolor, setcolor, setrgbpalette*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int color, maxcolor, ht;
   int y = 10;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   maxcolor = getmaxcolor();
   ht = 2 * textheight("W");

   /* display the default colors */
   for (color=1; color<=maxcolor; color++)
   {
      setcolor(color);
      sprintf(msg, "Color: %d", color);
      outtextxy(1, y, msg);
      y += ht;
   }
```

```
   /* wait for a key */
   getch();

   /* black out the colors one by one */
   for (color=1; color<=maxcolor; color++)
   {
      setpalette(color, BLACK);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

setrgbpalette

Funktion

Ändert einen Eintrag der aktuellen Farbpalette für den IBM-Adapter 8514.

Syntax

```
#include <graphics.h>
void far setrgbpalette(int colornum, int red, int green,
                       int blue);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

setrgbpalette kann für VGA-Treiber und den IBM 8514 verwendet werden. Für den IBM 8514-Adapter und für die VGA-Treiber im Modus mit 256 Farben liegt der gültige Bereich von *colornum* zwischen 0 und 255, für die übrigen VGA-Modi liegt er zwischen 0 und 15. Bei den Parametern *red*, *green* und *blue* werden nur das niederwertige Byte und bei diesem nur die sechs höchstwertigen Bits interpretiert.

colornum legt die Nummer des Farbeintrags fest, dessen Wert sich aus den Komponenten *red*, *green* und *blue* zusammensetzt.

Hinweis: Der Treiber IBM8514.BGI initialisiert die ersten 16 Paletteneinträge des IBM8514-Adapters mit Werten, die den Standardfarben einer EGA- bzw. VGA-Karte entsprechen und sorgt so dafür, daß »normale« Programme auch mit diesem Adapter lauffähig sind.

Rückgabewert

Keiner.

Siehe auch

setpalette

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
```

```
#include <conio.h>

int main(void)
{
   /* select a driver and mode that supports the use */
   /* of the setrgbpalette function.                 */
   int gdriver = VGA, gmode = VGAHI, errorcode;
   struct palettetype pal;
   int i, ht, y, xmax;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* grab a copy of the palette */
   getpalette(&pal);

   /* create gray scale */
   for (i=0; i<pal.size; i++)
      setrgbpalette(pal.colors[i], i*4, i*4, i*4);

   /* display the gray scale */
   ht = getmaxy() / 16;
   xmax = getmaxx();
   y = 0;
   for (i=0; i<pal.size; i++)
   {
      setfillstyle(SOLID_FILL, i);
      bar(0, y, xmax, y+ht);
      y += ht;
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

settextjustify

Funktion

Legt die Ausrichtung von Textausgaben durch Grafikroutinen fest.

Syntax

```
#include <graphics.h>
void far settextjustify(int horiz, int vert);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

Die durch *settextjustify* gesetzten Parameter bestimmen, in welchem Verhältnis Textausgaben zur aktuellen Position (CP) horizontal und vertikal ausgerichtet werden.

Gemäß Vorgabe werden die Parameter LEFT_TEXT (horizontale Ausrichtung) und TOP_TEXT (vertikale Ausrichtung) gesetzt.

Für die Ausrichtung von Texten definiert der Aufzählungstyp *text_just* in graphics.h die folgenden Konstanten:

Name	Wert	Beschreibung
horizontal		
LEFT_TEXT	0	linksbündig
CENTER_TEXT	1	zentriert
RIGHT_TEXT	2	rechtsbündig
vertikal		
BOTTOM_TEXT	0	Ausrichtung an der Grundlinie
CENTER_TEXT	1	zentriert
TOP_TEXT	2	Ausrichtung am oberen Rand

Wenn *horiz* auf LEFT_TEXT und *direction* auf HORIZ_DIR gesetzt wurde, dann wird nach einem Aufruf von *outtext*(*string*) die aktuelle Position um *textwidth*(*string*) nach rechts verschoben.

settextjustify kann nicht zusammen mit Stream-Funktionen oder im Textmodus verwendet werden.

Rückgabewert

Keiner. Falls ein ungültiger Wert an *settextjustify* übergeben wurde, liefert *graphresult* -11 zurück; die Ausrichtung des Texts bleibt dann unverändert.

Siehe auch

gettextsettings, *graphresult*, *outtext*, *settextstyle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
/* function prototype */
void xat(int x, int y);

/* horizontal text justification settings */
char *hjust[] = { "LEFT_TEXT",
                  "CENTER_TEXT",
                  "RIGHT_TEXT"
                };

/* vertical text justification settings */
char *vjust[] = { "LEFT_TEXT",
                  "CENTER_TEXT",
                  "RIGHT_TEXT"
                };

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, hj, vj;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through text justifications */
   for (hj=LEFT_TEXT; hj<=RIGHT_TEXT; hj++)
      for (vj=LEFT_TEXT; vj<=RIGHT_TEXT; vj++)
      {
         cleardevice();
         /* set the text justification */
         settextjustify(hj, vj);

         /* create a message string */
         sprintf(msg, "%s  %s", hjust[hj], vjust[vj]);
         /* create cross hairs on the screen */
         xat(midx, midy);
```

```
            /* output the message */
            outtextxy(midx, midy, msg);
            getch();
          }
        /* clean up */
        closegraph();
        return 0;
}
/* draw an "x" at (x, y) */
void xat(int x, int y)
{
  line(x-4, y, x+4, y);
  line(x, y-4, x, y+4);
}
```

settextstyle

Funktion

Legt den Font, die Schreibrichtung und die Größe folgender Textausgaben im Grafikmodus fest.

Syntax

```
#include <graphics.h>
void far settextstyle(int font, int direction,
                      int charsize);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

settextstyle setzt einen Font (*font*), die Schreibrichtung (*direction*) und den Vergrößerungsfaktor (*charsize*). Ein Aufruf von *settextstyle* beeinflußt alle nachfolgenden Textausgaben mit *outtext* bzw. *outtexxy*.

font: Ein 8*8-Bitmap-Font und verschiedene Vektor-Fonts stehen zur Verfügung. Der Aufzählungstyp *font_names* definiert die folgenden Konstanten in graphics.h:

Name	Wert	Beschreibung
DEFAULT_FONT	0	8 * 8 Bitmap-Font (Standard)
TRIPLEX_FONT	1	Vektor-Font »Triplex«
SMALL_FONT	2	Kleiner Vektor-Font
SANSSERIF_FONT	3	Vektor-Font »Sans serif«
GOTHIC_FONT	4	Vektor-Font »Gothic«
SCRIPT_FONT	5	Vektor-Font »Script«
SIMPLEX_FONT	6	Vektor-Font »Simplex«
TRIPLEX_SCR_FONT	7	Vektor-Font »Triplex Script«
COMPLEX_FONT	8	Vektor-Font »Complex«
EUROPEAN_FONT	9	Vektor-Font »European«
BOLD_FONT	10	Vektor-Font »Bold«

Der vordefinierte Bitmap-Font (DEFAULT_FONT) ist Teil des Grafikpakets und immer verfügbar. Die anderen Fonts sind in separaten Dateien (*.CHR) gespeichert und können mit *settextstyle* entweder nur bei Bedarf einzeln von der Diskette geladen werden oder mit dem Zusatzprogramm BGIOBJ in .OBJ-Dateien konvertiert und mit *registerbgifont* »registriert« und so in das Programm gelinkt werden.

Der Wert von *direction* gibt an, ob Textausgaben von links nach rechts (d.h. horizontal) oder von unten nach oben (d.h. in vertikaler Richtung) erfolgen. Mögliche Werte für *direction* sind:

Name	Wert	Beschreibung
HORIZ_DIR	0	von links nach rechts (Standard)
VERT_DIR	1	Ausgabe von unten nach oben

charsize legt den Vergrößerungsfaktor (0 bis 10) für Textausgaben fest. Falls *charsize* ungleich Null ist, können sowohl Zeichen aus Vektor- als auch aus Bitmap-Fonts davon betroffen sein. Der Wert Null für *charsize* ist nur auf Vektor-Fonts anwendbar.

- Falls *charsize* gleich 1 ist, zeigen *outtext* und *outtextxy* Zeichen des 8*8 Bitmap-Fonts in einem Rechteck mit 8*8 Pixeln am Bildschirm an.

- Falls *charsize* gleich 2 ist, zeigen diese Ausgabefunktionen Zeichen des 8*8 Bitmap-Fonts in einem Rechteck mit 16*16 Pixeln (usw., bis zu einem Limit vom zehnfachen der normalen Größe) an.

- Falls *charsize* gleich 0 ist, vergrößern die Ausgabefunktionen *outtext* und *outtextxy* den Text bei Verwendung eines Vektor-Fonts mit dem Standardvergrößerungsfaktor (4) bzw. mit dem vom Benutzer mit *setusercharsize* definierten Vergrößerungsfaktor.

Mit *textheight* und *textwidth* können Sie jederzeit die tatsächlichen Abmessungen des Textes abrufen.

Rückgabewert Keiner.

Siehe auch *gettextsettings*, *graphresult*, *installuserfont*, *settextjustify*, *setusercharsize*, *textheight*, *textwidth*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the text styles supported */
char *fname[] = { "DEFAULT font",
```

```
                    "TRIPLEX font",
                    "SMALL font",
                    "SANS SERIF font",
                    "GOTHIC font"
                  };

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int style, midx, midy;
   int size = 1;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* loop through the available text styles */
   for (style=DEFAULT_FONT;
        style<=GOTHIC_FONT; style++)
   {
      cleardevice();
      if (style == TRIPLEX_FONT)
         size = 4;

      /* select the text style */
      settextstyle(style, HORIZ_DIR, size);

      /* output a message */
      outtextxy(midx, midy, fname[style]);
      getch();
   }
```

```
   /* clean up */
   closegraph();
   return 0;
}
```

setusercharsize

Funktion

Legt voneinander unabhängige Vergrößerungsfaktoren in X- und Y-Richtung für Vektor-Fonts fest.

Syntax

```
#include <graphics.h>
void far setusercharsize(int multx, int divx,
                         int multy, int divy);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

setusercharsize erlaubt im Gegensatz zu *settextstyle* für Vektor-Fonts voneinander unabhängige Vergrößerungsfaktoren in X- und Y-Richtung: die Breite eines Zeichens wird zuerst mit *multx* multipliziert und danach durch *divx* dividiert. Dasselbe gilt für die Zeichenhöhe und die Parameter *multy* bzw. *divy*.

Soll beispielsweise Text *ABC* doppelt so breit und 50 % höher als in der Standardvorgabe festgelegt ausgegeben werden, muß *setusercharsize* mit den Parametern

```
multx = 2;   divx = 1;
multy = 3;   divy = 2;
```

aufgerufen werden.

Die von *setusercharsize* gesetzten Werte sind nur dann aktiv, wenn *charsize* durch einen vorausgehenden Aufruf von *settextstyle* auf 0 gesetzt wurde.

Rückgabewert

Keiner.

Siehe auch

gettextsettings, graphresult, settextstyle

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
```

```
   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)      /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);     /* terminate with an error code */
   }

   /* select a text style */
   settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);

   /* move to the text starting position */
   moveto(0, getmaxy() / 2);

   /* output some normal text */
   outtext("Norm ");

   /* make the text 1/3 the normal width */
   setusercharsize(1, 3, 1, 1);
   outtext("Short ");

   /* make the text 3 times normal width */
   setusercharsize(3, 1, 1, 1);
   outtext("Wide");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setvbuf

Funktion Explizite Zuordnung eines Puffers zu einem Stream.

Syntax

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int type,
            size_t size);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *setvbuf* ordnet der durch *stream* angegebenen Datei den Puffer *buf* anstelle des automatisch vergebenen Puffers zu. Die Funktion kann nur angewendet werden, wenn die Datei bereits offen ist.

Mit der Angabe des Zeigerwertes NULL anstelle von *buf* wird über einen Aufruf von *malloc* ein Puffer der Größe *size* belegt

und der Datei zugeordnet. Der Puffer wird automatisch freigegeben, wenn die Datei wieder geschlossen wird.

Hinweis: size muß größer als 0 sein und kann einen maximalen Wert von 32767 haben.

stdin und *stdout* arbeiten ungepuffert, solange sie nicht umgeleitet sind. Umleitungen bewirken dagegen vollständige Pufferung.

Die »Pufferung« von Ein- und Ausgaben bedeutet: Ausgaben werden zeichenweise zwischengespeichert und blockweise geschrieben; Eingaben werden blockweise von der Datei bzw. dem Gerät gelesen und danach zeichenweise aus dem Puffer geliefert.

Der Parameter *type* von *setvbuf* muß den Wert einer der drei folgenden Konstanten enthalten, die in stdio.h definiert sind:

_IOFBF – Ein-/Ausgaben werden vollständig gepuffert. Wenn der Puffer »leergelesen« ist, versucht der nächste Eingabebefehl, den Puffer wieder komplett mit Daten zu füllen (liest also *size* Bytes). Ausgaben füllen den Puffer bis zum letzten Byte, erst danach wird der komplette Pufferinhalt geschrieben.

_IOLBF – Ausgaben werden zeilenweise gepuffert: der Pufferinhalt wird nach jeder Ausgabe eines Zeilenvorschubs physikalisch geschrieben. Eingaben arbeiten dagegen genauso wie bei vollständiger Pufferung.

_IONBF – Ein-/Ausgaben geschehen ohne das Dazwischenschalten eines Puffers, die Parameter *buf* und *size* werden von *setvbuf* ignoriert. Jeder Eingabebefehl liest direkt von der Datei, jeder Ausgabebefehl hat eine sofortige Schreibaktion zur Folge.

Hinweis: Für *buf* kann eine lokale Variable verwendet werden – allerdings sollte man in diesem Fall darauf achten, die Datei vor dem Ende der entsprechenden Funktion wieder zu schließen.

Rückgabewert

setvbuf liefert bei fehlerfreier Ausführung den Wert 0. Ungültige Werte für *type* oder *size*, fehlender Speicherplatz für *malloc* oder die Anwendung auf eine nicht geöffnete Datei erzeugen ein Funktionsergebnis ungleich Null.

Siehe auch

fflush, fopen, setbuf

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *input, *output;
   char bufr[512];

   input = fopen("file.in", "r+b");
   output = fopen("file.out", "w");
```

```
   /* set up input stream for minimal disk access,
      using our own character buffer */
   if (setvbuf(input, bufr, _IOFBF, 512) != 0)
      printf("failed to set up buffer for input "
             "file\n");
   else
      printf("buffer set up for input file\n");

   /* set up output stream for line buffering using
      space that will be obtained through an indirect
      call to malloc */
   if (setvbuf(output, NULL, _IOLBF, 132) != 0)
      printf("failed to set up buffer for output "
             "file\n");
   else
      printf("buffer set up for output file\n");

   /* perform file I/O here */

   /* close files */
   fclose(input);
   fclose(output);
   return 0;
}
```

setverify

Funktion — Legt fest, ob DOS geschriebene Daten zusätzlich überprüft oder nicht.

Syntax

```
#include <dos.h>
void setverify(int value);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — Über das Flag *Verify* wird festgelegt, ob DOS geschriebene Diskettensektoren automatisch überprüft oder nicht. Normalerweise ist dieses Flag auf *off* gesetzt (eine direkte Überprüfung findet also nicht statt).

setverify setzt das Flag. Für *value* sind nur zwei Werte zulässig:

0 Überprüfung aus
1 Überprüfung an

Rückgabewert — Keiner.

Siehe auch — *getverify*

Beispiel

```
#include <stdio.h>
#include <conio.h>
```

```
#include <dos.h>

int main(void)
{
   int verify_flag;

   printf("Enter 0 to set verify flag off\n");
   printf("Enter 1 to set verify flag on\n");

   verify_flag = getch() - 0;

   setverify(verify_flag);

   if (getverify())
      printf("DOS verify flag is on\n");
   else
      printf("DOS verify flag is off\n");

   return 0;
}
```

setviewport

Funktion

Setzt ein Zeichenfenster.

Syntax

```
#include <graphics.h>
void far setviewport(int left, int top, int right,
                     int bottom, int clip);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

setviewport legt mit (*left,top*) die obere linke Ecke und mit (*right, bottom*) die untere rechte Ecke eines neuen Zeichenfensters fest und arbeitet mit absoluten Koordinaten. Der Aufruf

```
setviewport(0,0,100,100,1);
```

setzt ein Zeichenfenster von 100 Pixeln Breite, das sich in der oberen linken Ecke des Bildschirms befindet – unabhängig davon, ob zuvor ein anderes Zeichenfenster gesetzt war oder nicht.

setviewport setzt die aktuelle Position (CP) auf den Punkt (0,0) innerhalb des neuen Fensters. Der »Fensterinhalt« wird durch diese Funktion nicht verändert.

Der Parameter *clip* legt fest, ob Zeichenaktionen am Rand des Fensters abgeschnitten werden sollen (*clip* = 1) oder nicht (*clip* = 0).

Rückgabewert

Keiner. Der Aufruf von *setviewport* mit Werten, die die Grenzen des Bildschirms überschreiten, läßt die aktuellen Grafikeinstel-

lungen unverändert. Ein nachfolgender Aufruf von *graphresult* liefert in diesem Fall -11.

Siehe auch *clearviewport, getviewsettings, graphresult*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define CLIP_ON 1   /* activates clipping in
                       viewport */

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   setcolor(getmaxcolor());

   /* message in default full-screen viewport */
   outtextxy(0, 0,
             "* <-- (0, 0) in default viewport");

   /* create a smaller viewport */
   setviewport(50, 50, getmaxx()-50, getmaxy()-50,
               CLIP_ON);

   /* display some text */
   outtextxy(0, 0,
             "* <-- (0, 0) in smaller viewport");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

setvisualpage

Funktion — Wählt eine von mehreren Grafikspeicherseiten zur Anzeige aus.

Syntax

```
#include <graphics.h>
void far setvisualpage(int page);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *setvisualpage* wählt die Grafikseite *page* zur Anzeige auf dem Bildschirm aus.

Rückgabewert — Keiner.

Siehe auch — *graphresult, setactivepage*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
  /* select a driver and mode that supports
     multiple pages. */
  int gdriver = EGA, gmode = EGAHI, errorcode;
  int x, y, ht;

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)  /* an error occurred */
  {
    printf("Graphics error: %s\n",
           grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
  }

  x = getmaxx() / 2;
  y = getmaxy() / 2;
  ht = textheight("W");

  /*  select the off screen page for drawing */
  setactivepage(1);
```

```
  /* draw a line on page #1 */
  line(0, 0, getmaxx(), getmaxy());

  /* output a message on page #1 */
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(x, y, "This is page #1:");
  outtextxy(x, y+ht, "Press any key to halt:");

  /* select drawing to page #0 */
  setactivepage(0);

  /* output a message  on page #0 */
  outtextxy(x, y, "This is page #0.");
  outtextxy(x, y+ht,
            "Press any key to view page #1:");
  getch();

  /* select page #1 as the visible page */
  setvisualpage(1);

  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

setwritemode

Funktion

Legt den Schreibmodus für Zeichenoperationen von Linien im Grafikmodus fest.

Syntax

```
#include <graphics.h>
void far setwritemode(int mode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

setwritemode legt fest, ob Zeichenaktionen mit *line, linerel, lineto, rectangle* und *drawpoly* den vorherigen Bildschirminhalt direkt überschreiben (COPY_PUT) oder ob dazu eine XOR-Operation verwendet wird. COPY_PUT verwendet die Assembler-Anweisung MOV und überschreibt daher den vorherigen Bildschirminhalt. Mit XOR_PUT läßt sich die gezeichnete Linie durch einen weiteren Zeichenbefehl im selben Modus ohne Zerstörung des Hintergrundes wieder vom Bildschirm entfernen.

Die beiden möglichen Parameter von *setwritemode* sind in graphics.h als Konstanten definiert:

Name	Wert	Beschreibung
COPY_PUT	0	Überschreiben durch direkte Kopie
XOR_PUT	1	Exklusives-ODER-Operation mit dem vorherigen Inhalt

Rückgabewert Keiner.

Siehe auch *drawpoly, line, linerel, lineto, putimage, rectangle*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   xmax = getmaxx();
   ymax = getmaxy();

   /* select XOR drawing mode */
   setwritemode(XOR_PUT);

   /* draw a line */
   line(0, 0, xmax, ymax);
   getch();

   /* erase the line by drawing over it */
   line(0, 0, xmax, ymax);
   getch();

   /* select overwrite drawing mode */
   setwritemode(COPY_PUT);

   /* draw a line */
   line(0, 0, xmax, ymax);
```

```
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

signal

Funktion Legt fest, wie das Programm auf Software-Signale reagiert.

Syntax
```
#include <signal.h>
void (*signal(int sig, void (*func)
            (int sig[,int subcode])))(int);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung Über die Funktion *signal* kann der Programmierer bestimmen, wie das Programm auf die Auslösung eines Software-Signals reagiert. Mit ihr lassen sich nicht nur die Einträge der »Aktionstabelle« verändern, sondern auch eigene Routinen installieren.

signal.h definiert drei Standardfunktionen zur Bearbeitung von Software-Signalen:

Funktionszeiger	Bedeutung
SIG_DFL	»signal default« – beendet das Programm
SIG_IGN	»signal ignore« – ignoriert das Signal, das Programm wird fortgesetzt
SIG_ERR	»signal error« – wird benutzt, um eine Fehlerbedingung bei der Ausführung von **signal** anzuzeigen (d.h. einen Fehler beim Setzen eines Eintrags)

Die aktuellen Signaltypen zusammen mit ihrer »Standardaktion« sind:

Signaltyp	Bedeutung
SIGARBRT	»Abnormal termination« – Standardaktion: *_exit(3)*
SIGFPE	»Arithmetic error«, ausgelöst von Divisionen durch Null, ungültigen Operationen, usw. Standardaktion: *_exit(1)*
SIGILL	»Illegal operation«, ausgelöst durch ungültige Operation. Standardaktion: *_exit(1)*
SIGINT	»interrupt«, ausgelöst durch die Eingabe von Strg C . Standardaktion: Aufruf des Interrupts 0x23

Signaltyp	Bedeutung
SIGSEGV	»Segment violation«, unzulässiger Speicherzugriff. Standardaktion: *_exit(1)*
SIGTERM	»Request for program termination« – Standardaktion: *_exit(1)*

Hinweis: signal.h definiert einen Datentyp *sig_atomic_t* als »Atom« eines Speicherzugriffs, der auch durch asynchrone Interrupts nicht unterbrochen werden kann. Für die Prozessoren der 8086-Familie ist dieses »Atom« ein Wort mit 16 Bit, d.h. ein **int**.

Wenn ein Software-Signal vorliegt (unabhängig davon, ob es durch eine »echte« Fehlerbedingung oder einen Aufruf von *raise* erzeugt wurde), dann prüft die Laufzeitbibliothek, ob für diesen Signaltyp ein Handler installiert wurde:

- wenn nicht, wird SIG_DFL aufgerufen und das Programm beendet;
- wenn ja, wird der »Aktionseintrag« für diesen Signaltyp auf SIG_DFL zurückgesetzt und der Handler mit dem Signaltyp als Parameter aufgerufen.

Benutzerdefinerte Handler können mit einem Aufruf von *abort*, *_exit*, *exit* oder *longjmp* enden – oder auch mit einem normalen Rücksprung zum unterbrochenen Programm.

Erweiterungen des ANSI-C-Standards
Borlands C++ Compiler implementieren die Signaltypen SIGFPE, SIGSEGV und SIGILL mit Erweiterungen: die entsprechenden Handler bekommen nicht nur den Signaltyp, sondern auch noch einen oder zwei zusätzliche Parameter übergeben.

Wenn diese drei Signaltypen über *raise* ausgelöst werden, erhält der Handler einen der in folgender Tabelle aufgeführten Werte als zusätzlichen Parameter (und kann daran erkennen, daß kein »echter« Ausnahmezustand vorliegt):

Die Werte der folgenden Konstanten sind in float.h deklariert.

Signaltyp	2. Parameter
SIGFPE	FPE_EXPLICITGEN
SIGSEGV	SEGV_EXPLICITGEN
SIGILL	ILL_EXPLICITGEN

Wird ein Signal des Typs SIGFPE als Folge eines 8087-Fehlers ausgelöst, bekommt der Handler die Art dieses Fehlers als zweiten Parameter übergeben; die dazugehörigen Konstanten (FPE_...) sind ebenfalls in float.h deklariert.

Für die Signaltypen SIGSEGV und SIGILL oder den Typ SIGFPE als Folge eines 8086-Fehlers erhält der Handler zwei zusätzliche Parameter:

1. Den Signaltyp entsprechend dem ANSI-C-Standard (also SIGSEGGV, SIGILL oder SIGFPE).

2. Einen Integer-Zeiger auf den Stack der *interrupt*-Routine, die der Handler als Folge des Signals aufgerufen hat. Auf dem Stack dieser Routine sind die Register in derselben Reihenfolge wie die Parameter der Interrupt-Funktion (BP, DI, SI, DS, ES, DX, CX, BX, AX, IP, CS, FLAGS) gespeichert. Wenn die Behandlungsroutine beispielsweise als

```
void Handler(int sig, int type, int *reglist);
```

deklariert ist, dann kann sie mit

```
*(reglist +2) = new_DI_value
```

festlegen, welchen Wert das Register DI bei der Rückkehr zum unterbrochenen Programm haben soll (vgl. Beispiel 2).

Fehlerarten für SIGFPE
Für Signale des Typs SIGFPE sind in float.h folgende Konstanten definiert. Sie entsprechen den Fehlern, die der 8087-Prozessor erkennen kann, sowie INTEGER DIVIDE BY ZERO und INTERRUPT ON OVERFLOW beim Hauptprozessor.

Konstante	Bedeutung
FPE_INTOVFLOW	INTO-Befehl mit gesetztem OF-Flag
FPE_INTDIV0	Integerdivision durch Null
FPE_INVALID	Ungültiger 8087-Befehl
FPE_ZERODIVIDE	Division durch Null
FPE_OVERFLOW	Überlauf
FPE_UNDERFLOW	Unterlauf
FPE_INEXACT	Ergebnis ist ungenau (Rundungsfehler)
FPE_EXPLICITGEN	Signal durch *raise*(*SIGFPE*) ausgelöst

Hinweis: FPE_INTOVFLOW und FPE_INTDIV0 werden durch Integer-Operationen ausgelöst, die restlichen Signale durch Gleitkommaoperationen, d.h. entweder direkt durch den 8087-Prozessor oder durch die Routinen des 8087-Emulators. Ob ein Ausnahmezustand bei Gleitkommaoperationen ein Signal setzt, hängt vom Steuer-WORD des Coprozessors (bzw. des Emulators) ab, das über die Routine *_control87* verändert werden kann.

Hinweis: Denormalisierte Werte und der dazugehörige Ausnahmezustand (*DeNorm*) werden direkt von der Laufzeitbibliothek behandelt und nicht an den Handler übergeben.

Fehlerarten für SIGSEGV
Für Signale des Typs SEGSEGV sind die folgenden Konstanten in signal.h definiert:

Konstante	Bedeutung
SEGV_BOUND	ausgelöst durch BOUND-Befehl
SEGV_EXPLICITGEN	ausgelöst durch *raise*(*SIGSEGV*)

Der Prozessorbefehl BOUND ist für die Prozessortypen 8088/86 nicht definiert, wird in der Laufzeitbibliothek nicht verwendet und auch nicht von Borlands C++ Compilern erzeugt – das Signal SIGSEGV kann also, außer durch einen Aufruf von *raise*, nur über *asm*-Anweisungen oder in ein Programm eingebundene Maschinenroutinen ausgelöst werden.

Fehlerarten für SIGILL
Für Signale des Typs SIGILL sind die folgenden Konstanten in signal.h definiert:

Konstante	Bedeutung
ILL_EXECUTION	ausgelöst durch den Prozessor – illegale Operationsanforderung
ILL_EXPLICITGEN	ausgelöst durch *raise*(*SIGILL*)

Die Prozessortypen 8088/86, NEC V20 und V30 lösen bei illegalen Befehlen keinen Interrupt aus, die Prozessortypen 186, 286, 386, NECV40 und NECV50 dagegen schon – das Signal SIGILL kann also, außer durch einen Aufruf von *raise*, nur auf Computern der AT-Klasse (und darüber) auftreten.

Das Ende einer Behandlungsroutine für Signale
Selbstdefinerte Behandlungsroutinen können mit einem Aufruf von *abort*, *_exit*, *exit* oder *longjmp* enden – oder auch mit einem normalen Rücksprung zum unterbrochenen Programm.

Für die Signaltypen SIGFPE, SIGSEGV und SIGILL empfiehlt sich ein normaler Rücksprung im allgemeinen nicht, denn das Programm hätte danach mit einem Durcheinander auf dem 8087, falschen Ergebnissen einer Division oder gar undefinierten Opcodes zu kämpfen. Generell sollte eine Behandlungsroutine deshalb eine Fehlermeldung ausgeben und das Programm danach beenden. Zu dieser Regel gibt es allerdings zwei Ausnahmen: wenn das Signal

- entweder über *raise* erzeugt wurde (das heißt, wenn die Behandlungsroutine als zweiten Parameter einen der Werte FPE_EXCPLICITGEN, SEGV_EXPLICITGEN usw. erhält) oder
- der Handler aktiv eingreift (d.h. Register des 8086 über Zugriffe auf den *interrupt*-Stack verändert werden bzw. ein *longjmp* ausführt wird),

dann könnte das Programm davon ausgehen, daß die Fehlerbedingung vollständig beseitigt wurde. Ein Abbruch wäre in diesen beiden Fällen also nicht zwingend.

Rückgabewert Bei fehlerfreier Ausführung liefert *signal* einen Zeiger auf die Routine, die zuvor als Handler für Signale des angegebenen Typs gesetzt war – normalerweise SIG_DFL). Im Falle eines Fehlers ist das Ergebnis von *signal* der Zeiger SIG_ERR, *errno* bekommt den Wert EINVAL.

Siehe auch *abort*, *_control87*, *ctrlbrk*, *exit*, *longjmp*, *raise*, *setjmp*

Beispiel

```
/* This example installs a signal handler routine for
   SIGFPE, catches an integer overflow condition,
   makes an adjustment to AX register, and returns
   This example program MAY cause your computer
   to crash, and will produce runtime errors
   depending on which memory model is used.
*/

#pragma inline
#include <stdio.h>
#include <signal.h>

#ifdef __cplusplus
   typedef void (*fptr)(int);
#else
   typedef void (*fptr)();
#endif

void Catcher(int *reglist)
{
   printf("Caught it!\n"); *(reglist + 8) = 3;
          /* make return AX = 3 */
}

int main(void)
{
   signal(SIGFPE, (fptr)Catcher);
        /* cast Catcher to appropriate type */
   asm     mov     ax,07FFFH     /* AX = 32767 */
   asm     inc     ax            /* cause overflow */
   asm     into                  /* activate handler */

   /* The handler set AX to 3 on return. If that
      hadn't happened, there would have been another
      exception when the next 'into' was executed
      after the 'dec' instruction. */
   asm     dec     ax          /* no overflow now */
   asm     into                /* doesn't activate */
   return 0;
}
```

sin, sinl

Funktion Liefert den Sinus des Arguments zurück.

Syntax

reelle Versionen:
```
#include <math.h>
double sin(double x);
long double sinl(long double x);
```

komplexe Version:
```
#include <complex.h>
complex sin(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*sinl*
x	x	x	x		*reelles sin*
x		x		x	*komplexes sin*

Beschreibung *sin* erwartet einen Winkel in der Einheit *rad* als Argument und liefert den dazugehörigen Sinus zurück.

sinl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Die Fehlerbehandlung kann für diese Funktionen über *matherr* bzw. *_matherrl* modifiziert werden.

Rückgabewert *sin* und *sinl* liefern den Sinus des Eingabewertes, also Werte im Bereich von -1 bis +1.

Der komplexe Sinus ist definiert als

$$sin(z) = (exp(i * z) - exp(-i * z)) / (2\ i)$$

Siehe auch *acos, asin, atan, atan2, complex, cos, tan*

Beispiel
```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x = 0.5;

   result = sin(x);
   printf("The sin() of %lf is %lf\n", x, result);
   return 0;
}
```

sinh, sinhl

Funktion Liefert den Sinus hyperbolicus des Arguments zurück.

Syntax

reelle Versionen:
```
#include <math.h>
double sinh(double x);
long double sinhl(long double x);
```

komplexe Version:
```
#include <complex.h>
complex sinh(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*sinhl*
x	x	x	x		*reelles sinh*
x		x		x	*komplexes sinh*

Beschreibung *sinh* berechnet den Sinus hyperbolicus. Dieser ist definiert durch $(e^{x} - e^{-x}) / 2$

sinhl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Die komplexe Version von *sin* ist definiert als

$$sinh(z) = (exp(z) - exp(-z)) / 2$$

Die Behandlung von Fehlern kann für diese Funktionen über *matherr* bzw. *_matherrl* modifiziert werden.

Rückgabewert Bei fehlerfreier Ausführung liefern die Funktionen den Sinus hyperbolicus des Arguments. Bei Überläufen wird HUGE_VAL bzw. _LHUGE_VAL (mit korrektem Vorzeichen) zurückgeliefert und die globale Variable *errno* bekommt den Wert:

ERANGE Ergebnis außerhalb des Rechenbereichs

Siehe auch *acos*, *asin*, *atan*, *atan2*, *complex*, *cos*, *cosh*, *sin*, *tan*, *tanh*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x = 0.5;

   result = sinh(x);
   printf("The hyperbolic sin() of %lf is %lf\n",
          x, result);
   return 0;
}
```

sleep

Funktion Hält die Ausführung des Programms für einen bestimmten Zeitraum an.

Syntax

```
#include <dos.h>
void sleep(unsigned seconds);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x			

Beschreibung *sleep* hält die Ausführung für die durch *seconds* angegebene Anzahl von Sekunden an und verwendet dabei die Uhr des Systems.

Die Genauigkeit der Routine ist von der Auswertung des »Clock Tick« durch DOS abhängig (d. h., daß die tatsächlichen Verzögerungszeiten maximal um 1/18 Sekunde schwanken können).

Rückgabewert Keiner.

Siehe auch *delay*

Beispiel
```
#include <dos.h>
#include <stdio.h>

int main(void)
{
   int i;

   for (i=1; i<5; i++)
   {
      printf("Sleeping for %d seconds\n", i);
      sleep(i);
   }
   return 0;
}
```

sopen

Funktion Öffnet eine Datei im Modus SHARED.

Syntax
```
#include <fcntl.h>
#include <sys\stat.h>
#include <share.h>
#include <io.h>
int sopen(char *path, int access, int shflag,
          [,int mode]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *sopen* öffnet die durch *path* angegebene Datei im Modus SHARED und bereitet – je nach den für *access, shflag* und *mode* angegebenen Werten – folgende Lese- und/oder Schreibaktionen vor. *sopen* ist definiert als:

```
open(path, (access) | (shflag), mode)
```

Für jeden Aufruf von *sopen* darf nur eines der Flags aus der ersten der beiden folgenden Listen angegeben sein. Die weite-

ren Flags können in beliebigen logischen Kombinationen verwendet werden. Die Flags werden über eine ODER-Operation miteinander verbunden.

1. Flags für Lesen und/oder Schreiben

O_RDONLY	Nur Lesen
O_WRONLY	Nur Schreiben
O_RDWR	Lesen und Schreiben

2. Weitere Zugriff-Flags

O_NDELAY	nur aus Gründen der UNIX-Kompatibilität definiert, unter DOS nicht verwendet.
O_APPEND	neu geschriebene Daten werden grundsätzlich angehängt, das Überschreiben bereits existierender Daten ist nicht möglich.
O_CREAT	falls die Datei bereits existiert, hat dieses Flag keine Wirkung. Wenn die Datei noch nicht existiert, wird sie erzeugt und die Bits von mode werden benutzt, um die Dateiattribute wie in *chmod* zu setzen.
O_TRUNC	wenn die Datei bereits existiert, wird ihr vorheriger Inhalt gelöscht, die Dateiattribute bleiben unverändert.
O_EXCL	nur zusammen mit O_CREAT. Falls die Datei bereits existiert, wird ein Fehler zurückgeliefert.
O_BINARY	legt den Modus *binary* fest, gelesene/geschriebene Daten werden nicht interpretiert.
O_TEXT	legt den Modus *text* fest .

Die symbolischen Konstanten O_... sind in fcntl.h definiert.

Wenn weder O_BINARY noch O_TEXT angegeben ist, wird die Datei in dem Modus, der durch die globale Variable *_fmode* gesetzt ist, geöffnet.

Bei der Verwendung von O_CREAT (d.h. der Erzeugung einer neuen Datei) bei der Bildung von *access* muß das Argument *mode* mit den folgenden symbolischen Konstanten festgelegt werden, die in sys\stat.h definiert sind:

S_IWRITE	Schreiben erlaubt
S_IREAD	Lesen erlaubt
S_IREAD | S_IWRITE	Lesen und Schreiben erlaubt

shflag legt den Typ des gemeinsamen Zugriffs, der auf die Datei *path* angewendet werden soll, fest. Die symbolischen Konstanten für *shflag* sind in share.h festgelegt.

SH_COMPAT	Setzt Kompatibilitätsmodus
SH_DENYRW	Lesen und Schreiben nicht erlaubt

SH_DENYWR	Schreiben nicht erlaubt
SH_DENYRD	Lesen nicht erlaubt
SH_DENYNONE	Lesen und Schreiben erlaubt
SH_DENYNO	Lesen und Schreiben erlaubt

Rückgabewert Wenn die Datei fehlerfrei geöffnet werden konnte, liefert die Funktion das Handle dieser Datei zurück (einen **int** mit einem Wert größer Null, über den weitere Operationen mit dieser Datei ausgeführt werden). Der Dateizeiger wird auf den Dateianfang gesetzt.

Im Fehlerfall liefert die Funktion den Wert -1 und setzt die globale Variable *errno* auf einen der folgenden Werte:

ENOENT	Suchweg oder Datei nicht vorhanden
EMFILE	Zu viele Dateien bereits offen
EACCES	Zugriff nicht gestattet
EINVACC	Ungültiger Zugriffscode

Siehe auch *chmod, close, creat, lock, lseek, _open, open, unlock, umask*

Beispiel

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
   int handle;
   int status;

   handle = sopen("c:\\autoexec.bat", O_RDONLY,
                  SH_DENYNO, S_IREAD);

   if (!handle)
   {
      printf("sopen failed\n");
      exit(1);
   }

   status = access("c:\\autoexec.bat", 6);
   if (status == 0)
      printf("read/write access allowed\n");
   else
      printf("read/write access not allowed\n");

   close(handle);
   return 0;
```

sound

Funktion — Aktiviert den eingebauten Lautsprecher.

Syntax

```
#include <dos.h>
void sound(unsigned frequency);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *sound* erzeugt einen Ton im eingebauten Lautsprecher, dessen Frequenz über den Wert von *frequency* in Hertz (Schwingungen pro Sekunde) festgelegt ist. Ein mit *sound* gestarteter Ton wird so lange erzeugt, bis der Lautsprecher durch einen Aufruf von *nosound* wieder abgeschaltet wird.

Rückgabewert — Keiner.

Siehe auche — *delay, nosound*

Beispiel

```
/* Emits a 440-Hz tone (A above middle C)
   for 1 second. */

#include <dos.h>

int main(void)
{
   sound(440);
   delay(1000);
   nosound();
   return 0;
}
```

spawn...

Funktion — Erzeugung und Ausführung von untergeordnten Prozessen.

Syntax

```
#include <process.h>
#include <stdio.h>
int spawnl(int mode, char *path, char *arg0,
           /* arg1...argn, NULL */);
int spawnle(int mode, char *path, char *arg0,
            /* arg1...argn, NULL, char envp[] */);
int spawnlp(int mode, char *path, char *arg0,
            /* arg1...argn, NULL */);
int spawnlpe(int mode, char *path, char *arg0,
             /* arg1...argn, NULL, char envp[] */);
int spawnv(int mode, char *path, char *argv[]);
int spawnve(int mode, char *path, char *argv[],
            char *envp[]);
```

```
int spawnvp(int mode, char *path, char *argv[]);
int spawnvpe(int mode, char *path, char *argv[],
             char *envp[]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

Die Funktionen der *spawn...*-Familie erzeugen und führen andere Programme aus, die auch untergeordnete Prozesse genannt werden. Es muß ausreichend Speicherplatz für das Laden und Ausführen dieser Prozesse vorhanden sein.

Das aufrufende Programm (*parent*) bleibt während der Ausführung des neu gestarteten Prozesses (*child*) im Speicher und wird nach dem Ablauf des untergeordneten Prozesses fortgesetzt.

Der Wert des Parameters *mode* legt fest, wie sich der übergeordnete Prozeß während der Ausführung des untergeordneten Prozesses verhält. Die folgenden Konstanten sind in process.h definiert:

P_WAIT – Der übergeordnete Prozess bleibt im Speicher, wird aber für die Dauer der Ausführung des untergeordneten Prozesses »stillgelegt«.

P_NOWAIT – Über- und untergeordnete Prozesse werden konkurrierend ausgeführt.

P_OVERLAY – Der übergeordnete Prozeß wird vom untegeordneten überschrieben und nach dessen Beendigung nicht fortgesetzt (derselbe Ablauf wie bei *exec...*).

Hinweis: P_NOWAIT ist gegenwärtig nicht verfügbar; die Verwendung erzeugt einen Fehler.

Der Parameter *path* gibt den Namen des untergeordneten Prozesses an. Verwendet wird der Standard-Suchalgorithmus von DOS:

- Wenn weder eine Namenserweiterung noch ein Punkt angegeben ist, wird zuerst nach Dateien ohne Erweiterung, dann nach .COM-Dateien und anschließend nach .EXE-Dateien gesucht.
- Wenn eine Namenserweiterung oder ein Punkt angegeben ist, wird nur nach dem angegebenen Dateinamen gesucht. Bei der Angabe xxx. sucht DOS nach der Datei XXX, bei der Angabe xxx.yyy nach der Datei XXX.YYY.
- Die »*p*«-Varianten von *spawn...* suchen zuerst im aktuellen Verzeichnis, danach in sämtlichen Verzeichnissen, die durch PATH festgelegt sind. Die Varianten ohne »*p*« suchen nur innerhalb des aktuellen Verzeichnisses.

Die einzelnen Funktionen der *spawn...*-Familie unterscheiden sich durch die Namensendungen *l*, *v*, *p* und *e*, wobei jeder dieser Buchstaben für andere Fähigkeiten steht. Es bedeuten:

p PATH – die *spawn*-Funktion sucht nach dem untergeordneten Prozeß nicht nur innerhalb des aktuellen Verzeichnisses, sondern auch in sämtlichen durch PATH festgelegten Verzeichnissen.

l LIST – die Kommandozeilenparameter werden an den untergeordneten Prozeß in Form einer Liste einzelner Argumente übergeben, also als *arg0, arg1, ... argn*. Die Listenform wird normalerweise verwendet, wenn die Zahl der Parameter bekannt (und konstant) ist.

v VARIABLE – die Kommandozeilen-Parameter werden an den untergeordneten Prozeß in Form von Zeiger-Arrays übergeben, d.h. als *argv[0], argv[1]...argv[n]*. Diese Form wird normalerweise bei variablen Parameterzahlen verwendet.

e ENVIRONMENT – erlaubt die Übergabe des Zeiger-Array *envp* (d.h. eines eigenen *Environment* für den untergeordneten Prozeß). Ein über die *spawn...*-Varianten ohne »e« aufgerufener untergeordneter Prozeß verwendet dieselben Umgebungsparameter wie das aufrufende Programm.

Jede Funktion der *spawn...*-Familie enthält entweder ein *l* oder ein *v* im Namen, die weiteren Kombinationen mit *p* und *e* sind optional. Zwei Beispiele dazu:

spawnl verwendet eine Liste von Argumenten, sucht nur das aktuelle Verzeichnis nach dem untergeordneten Prozeß ab und übergibt die *Environment*-Parameter, die auch für das aktuell laufende Programm gültig sind.

spawnvpe übergibt die Kommandozeilenparameter als Zeiger-Array, führt nötigenfalls eine Suche nach dem untergeordneten Prozeß über die mit PATH gesetzten Suchwege aus und übergibt *Environment*-Parameter, die vom Programmierer definiert wurden.

Sämtliche *spawn...*-Funktionen müssen mindestens ein Argument an den untergeordneten Prozeß übergeben, nämlich *arg0* bzw. *argv[0]*. Das erste Argument enthält per Konvention den Pfadnamen des untergeordneten Prozesses. Die Übergabe anderer Daten als erstem Argument erzeugt jedoch keinen Fehler. Zur Übergabe einer leeren Argumentliste an den untergeordneten Prozeß muß *arg0* bzw. *argv[0]* NULL sein.

Der *path*-Wert von *arg0* bzw. *argv[0]* ist nur unter den DOS-Versionen ab 3.0 für den untergeordneten Prozeß verfügbar. Bei Verwendung der »*l*«-Funktionen zeigt *arg0* normalerweise auf denselben String wie *path*, *arg1*,...,*argn* zeigen auf

weitere Strings, die Argumentenliste wird durch ein (obligatorisches!) NULL beendet.

Die »*e*«-Funktionen erwarten ein Zeiger-Array als Argument, dessen Inhalt als **envp*[] übergeben wird. Jedes Element dieses Array ist ein nullterminierter String der Form

```
envvar = Wert
```

wobei *envvar* der Name der *Environment*-Variablen und *Wert* der String-Wert ist, auf den *envvar* gesetzt wird. Das letzte Element des Array muß einen Zeiger mit dem Wert NULL enthalten. Dadurch wird das Listenende gekennzeichnet. Wenn *env*[*0*] den Wert NULL hat, verwendet der untergeordnete Prozeß die *Environment*-Tabelle des übergeordneten Prozesses.

Die Gesamtlänge sämtlicher als *arg* übergebener Kommandozeilenparameter darf 127 Byte nicht überschreiten (inklusive der automatisch eingesetzten Leerzeichen zur Trennung einzelner Parameter). Nullzeichen werden bei dieser Zählung nicht berücksichtigt.

spawn... schließt keine Dateien – durch den übergeordneten Prozeß geöffnete Dateien stehen auch dem untergeordneten Prozeß zur Verfügung.

Rückgabewert

Die fehlerfreie Ausführung einer *spawn...*-Funktion hat den Exit-Code des untergeordneten Prozesses als Rückgabewert. Der Exit-Code eines Programms kann über *exit*, *_exit*, *abort* usw. spezifisch festgelegt werden – normalerweise steht der Wert 0 für »erfolgreich ausgeführt«. Konnte die *spawn...*-Funktion dagegen nicht ausgeführt werden, dann wird -1 zurückgeliefert, und *errno* bekommt einen der folgenden Werte:

E2BIG	Argumentenliste zu lang
EINVAL	Ungültiger Wert für *mode*
ENOENT	Suchweg/Dateiname nicht gefunden
ENOEXEC	*exec*-Formatfehler
ENOMEM	Nicht genug Platz im Hauptspeicher

Siehe auch

abort, *atexit*, *_exit*, *exit*, *exec...*, *_fpreset*, *searchpath*, *system*

Beispiel

```
/* spawnl() and spawnle() example */

#include <process.h>
#include <stdio.h>
#include <conio.h>

void spawnl_example(void)
{
   int result;

   clrscr();
   result = spawnl(P_WAIT, "bcc.exe", NULL);
   if (result == -1)
```

```
   {
      perror("Error from spawnl");
      exit(1);
   }
}

void spawnle_example(void)
{
   int result;

   clrscr();
   result = spawnle(P_WAIT, "bcc.exe", NULL, NULL);
   if (result == -1)
   {
      perror("Error from spawnle");
      exit(1);
   }
}

int main(void)
{
spawnl_example();
spawnle_example();
}
```

_splitpath

Funktion

Zerlegt einen vollständigen Pfadnamen in Einzelkomponenten.

Syntax

```
#include <stdlib.h>
void _splitpath(const char *path, char *drive,
                char *dir, char *name, char *ext);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_splitpath übernimmt einen vollständigen Dateinamen (Parameter *path*) als String in der Form

```
X:\DIR\SUBDIR\NAME.EXT
```

und zerlegt *path* in seine vier Einzelkomponenten. Diese werden in den durch *drive*, *dir*, *name* und *ext* bezeichneten Strings gespeichert. Alle fünf Parameter müssen übergeben werden, einzelne davon dürfen aber NULL sein, was zur Folge hat, daß die zugehörige Komponente zwar analysiert, aber nicht gespeichert wird. Die Maximalgrößen aller beteiligen Strings sind in stdlib.h als Konstanten mit folgenden Werten definiert, wobei jede Größe Platz für das terminierende Nullzeichen beinhaltet:

Konstante	max.	String
_MAX_PATH	80	*path*
_MAX_DRIVE	3	*drive* inklusive Doppelpunkt (:)
_MAX_DIR	66	*dir* inklusive führendem und abschliessendem umgekehrten Schrägstrich (\)
_MAX_FNAME	9	*name*
_MAX_EXT	5	*ext* inklusive führendem Punkt (.)

_splitpath unterstellt, daß die durch *drive*, *dir*, *name* und *ext* bezeichneten Strings eine ausreichende Größe haben, um jede Nicht-Null-Komponente zu speichern.

_splitpath verteilt die Interpunktionszeichen aus *path* in folgender Weise:

drive erhält den Laufwerksbezeichner mit einem Doppelpunkt (C:, A:, usw.)

dir erhält einen führenden und abschließenden umgekehrten Schrägstrich (\BC\include\, \source\, usw.);

name enthält den Dateinamen;

ext enthält den Punkt, der die Namenserweiterung einleitet (.C, .EXE, usw.)

_makepath und *_splitpath* verhalten sich invers. Die Anwendung von *_makepath* auf die durch *_splitpath* ermittelten Elemente eines vollständigen Pfadnamens ergeben wieder das Original.

Rückgabewert Keiner.

Siehe auch *_fullpath, _makepath*

Beispiel

```
#include <dir.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char s[_MAX_PATH];
  char drive[_MAX_DRIVE];
  char dir[_MAX_DIR];
  char file[_MAX_FNAME];
  char ext[_MAX_EXT];

  /* get current working directory */
  getcwd(s,_MAX_PATH);
  if (s[strlen(s)-1] != '\\')

  /* append a trailing \ character */
    strcat(s,"\\");
  /* split the string to separate elems */
  _splitpath(s,drive,dir,file,ext);
```

```
  strcpy(file,"DATA");
  strcpy(ext,".TXT");

  /* merge everything into one string */
  _makepath(s,drive,dir,file,ext);

  /* display resulting string */
  puts(s);
  return 0;
}
```

sprintf

Funktion

Formatierte Ausgabe in einen String.

Syntax

```
#include <stdio.h>
int sprintf(char *buffer, const char *format,
            [,argument,...]);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung

sprintf »druckt« in formatierter Form in einen String und erwartet zumindest einen Formatstring (Parameter *format*), über den die Anzahl der Ausgaben und ihr Format festgelegt wird.

sprintf wendet die erste Formatangaben auf das erste Argument an, die zweite auf das zweite Argument usw. Die Anzahl der Formatangaben und Argumente muß gleich sein.

Hinweis: Weitere Details zu Formatangaben finden Sie unter *printf*.

Rückgabewert

sprintf liefert die Anzahl der ausgegebenen Zeichen zurück, wobei das abschließende Nullzeichen nicht mitgezählt wird. Im Falle eines Fehlers ist das Funktionsergebnis EOF.

Siehe auch

fprintf, printf

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   char buffer[80];

   sprintf(buffer, "An approximation of Pi is %f\n",
           M_PI);
   puts(buffer);
   return 0;
}
```

sqrt, sqrtl

Funktion Berechnet die positve Quadratwurzel des Arguments.

Syntax

reelle Versionen:

```
#include <math.h>
double sqrt(double x);
long double sqrtl(long double x);
```

komplexe Version:

```
#include <complex.h
complex sqrt(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*sqrtl*
x	x	x	x		*reelles sqrt*
x		x		x	*komplexes sqrt*

Beschreibung *sqrt* berechnet die positive Quadratwurzel des Arguments *x*.

sqrtl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Für komplexe Zahlen *x* liefert sqrt(x) die komplexe Wurzel, deren *arg* (Winkel der Zahl in der komplexen Zahlenebene) *arg(x)/2* beträgt.

Die komplexe Version von *sqrt* ist definiert als:

$$sqrt(z) = sqrt(abs(z))\ (cos(arg(z)\ /\ 2\ + i\ sin(arg(z)\ /\ 2))$$

Rückgabewert *sqrt* und *sqrtl* liefern die positive Quadratwurzel des Arguments, wenn *x* eine reelle Zahl und positiv ist. Ist *x* reell und negativ, wird die globale Variable *errno* auf den Wert

EDOM Bereichsfehler

gesetzt.

Die Behandlung von Fehlern kann für diese Funktionen über die Routinen *matherr* und *_matherrl* modifiziert werden.

Siehe auch *complex, exp, log, pow*

Beispiel

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double x = 4.0, result;

   result = sqrt(x);
   printf("The square root of %lf is %lf\n",
          x, result);
   return 0;
}
```

srand

Funktion — Setzt einen Startwert für die Erzeugung von »Zufallszahlen«.

Syntax

```
#include <stdlib.h>
void srand(unsigned seed);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — Der »Zufallszahlengenerator« wird beim Start des Programms mit dem Wert 1 initialisiert – über einen Aufruf von *srand* kann jederzeit ein neuer Startwert (*seed*) gesetzt werden.

Rückgabewert — Keiner.

Siehe auch — *rand, random, randomize*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(void)
{
   int i;
   time_t t;

   srand((unsigned) time(&t));
   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
       printf("%d\n", rand() % 100);
   return 0;
}
```

sscanf

Funktion — Liest und formatiert Eingaben aus einem String.

Syntax

```
#include <stdio.h>
int sscanf(const char *buffer,
           const char *format,[adress,...];
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *sscanf* liest eine Reihe von Eingabefeldern zeichenweise aus einem String. Auf jedes Feld wird dann die entsprechende Formatangabe angewendet, die *sscanf* über den Zeiger *format* auf den Formatstring übergeben wurde. Der Formatstring legt die Anzahl der Eingaben und ihr Format fest. Gelesene Ein-

gaben werden an der Adresse, die in dem *format* nachfolgenden Parameter übergeben wurde, gespeichert. Nach der Analyse der ersten Formatanweisung wird ein Eingabefeld gelesen, konvertiert und an der entsprechenden Adresse gespeichert. Die Anzahl von Eingabefeldern muß der Anzahl von Formatangaben und Adressen genau entsprechen.

sscanf kann unter Umständen das Lesen eines bestimmten Eingabefeldes, bevor zum Beispiel ein »Whitespace«-Zeichen erreicht wird, abbrechen oder das Lesen des Strings komplett beenden.

Hinweis: In der Beschreibung zu *scanf* finden Sie eine Erläuterung der möglichen Gründe. Außerdem werden dort die Formatangaben detailliert besprochen.

Rückgabewert

sscanf liefert die Anzahl der fehlerfrei gelesenen, konvertierten und gespeicherten Felder zurück. Wenn keine Zuweisungen erfolgt sind, hat das Ergebnis den Wert 0.

Der Versuch, über das Ende des Strings hinauszulesen, liefert den Wert EOF.

Siehe auch

fscanf, *scanf*

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

char *names[4] = {"Peter", "Mike", "Shea", "Jerry"};

#define NUMITEMS 4

int main(void)
{
   int   loop;
   char  temp[4][80];

   char  name[20];
   int   age;
   long  salary;

/* clear the screen */
   clrscr();

/* create name, age and salary data */
   for (loop=0; loop < NUMITEMS; ++loop)
      sprintf(temp[loop], "%s %d %ld", names[loop],
              random(10) + 20, random(5000) + 27500L);

/* print title bar */
   printf("%4s | %-20s | %5s | %15s\n", "#", "Name",
          "Age", "Salary");
   printf("   ------------------------------------"
          "------------\n");
```

```
/* input a name, age and salary data */
   for (loop=0; loop < NUMITEMS; ++loop)
      {
      sscanf(temp[loop],"%s %d %ld", &name, &age,
             &salary);
      printf("%4d | %-20s | %5d | %15ld\n",
             loop + 1, name, age, salary);
      }

   return 0;
}
```

stackavail

Funktion

Ruft die Größe des verfügbaren Speichers ab.

Syntax

```
#include <malloc.h>
size_t stackavail(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x			x	

Beschreibung

stackavail gibt die Größe des verfügbaren Speichers auf dem Stack in Bytes zurück. Diese Größe entspricht dem dynamischen Speicher, auf den *alloca* zugreifen kann.

Rückgabewert

stackavail liefert einen Wert vom Typ *size_t* zurück, der der Größe des verfügbaren Speichers in Bytes entspricht.

Siehe auch

alloca, allocmem, coreleft, _dos_allocmem, farcoreleft, stklen (globale Variable)

Beispiel

```
#include <malloc.h>
#include <stdio.h>

int main(void)
{
  char *buf;

  printf("\nThe stack: %u\tstack pointer: %u",
         stackavail(), _SP);
  buf = (char *) alloca(100 * sizeof(char));
  printf("\nNow, the stack: %u\tstack pointer: %u",
         stackavail(), _SP);
  return(0);
}
```

Programmausgabe:

```
The stack: 63258   stack pointer: 65524
Now, the stack: 63424    stack pointer: 65420
```

_status87

Funktion Liefert den Status der Gleitkommafunktionen zurück.

Syntax
```
#include <float.h>
unsigned int _status87(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Diese Funktion liefert den aktuellen Wert des Status-WORD der Gleitkommafunktionen zurück. Das WORD muß als Bitmuster interpretiert werden und enthält neben den Statusbits des Coprozessors verschiedene Flags für Ausnahmezustände (Überläufe, Divisionen durch Null etc).

Eine Beschreibung der entsprechenden Bit-Konstanten finden Sie in der Deklarationsdatei float.h.

Rückgabewert *_status87* liefert das Status-WORD des Gleitkommapakets als Bitmuster zurück.

Beispiel
```
#include <stdio.h>
#include <float.h>

int main(void)
{
   float x;
   double y = 1.5e-100;

   printf("Status 87 before error: %x\n",_status87());

   x = y;  /* <-- force an error to occur */
   y = x;

   printf("Status 87 after error : %x\n",_status87());
   return 0;
}
```

stime

Funktion Setzt Datum und Uhrzeit des Systems.

Syntax
```
#include <time.h>
int stime(time_t *tp);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *stime* setzt Datum und Uhrzeit des Systems auf den über den Zeiger *tp* übergebenen Wert, der in Sekunden seit dem 1.

Januar 1970, 0 Uhr 00:00 (Greenwich Mean Time) gemessen wird.

Rückgabewert *stime* liefert immer den Wert 0.

Siehe auch *asctime*, *ftime*, *gettime*, *gmtime*, *localtime*, *time*, *tzset*

Beispiel

```
#include <stdio.h>
#include <time.h>

int main(void)
{
  time_t t;

  t = time(NULL);

  printf("Current date is %s", ctime(&t));

  t -= 24L*60L*60L;  /* Back up to same time previous
day */

  stime(&t);
  printf("\nNew date is %s", ctime(&t));

  return 0;
}
```

stpcpy

Funktion Kopiert einen String in einen anderen.

Syntax

```
#include <string.h>
char *stpcpy(char *dest, const char *src);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *stpcpy* kopiert den Inhalt des durch *src* angegebenen nullterminierten Strings in den durch *dest* angegebenen Speicherbereich. Der Vorgang wird abgebrochen, sobald das abschließende Nullzeichen erreicht ist.

Rückgabewert *stpcpy* liefert einen Zeiger zurück, der auf das neue Ende von *dest* zeigt, also *dest* + *strlen*(*src*).

Siehe auch *strcpy*

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
```

```
{
   char string[10];
   char *str1 = "abcdefghi";

   stpcpy(string, str1);
   printf("%s\n", string);
   return 0;
}
```

strcat, _fstrcat

Funktion Hängt einen String an einen anderen.

Syntax
```
#include <string.h>
```

Near-Version:
```
char *strcat(char *dest, const char *src);
```

Far-Version:
```
char far * far _fstrcat(char far *dest,
                        const char far *src);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strcat* hängt sämtliche Zeichen von *src* an die bereits in *dest* vorhandenen Zeichen an. Die Länge des resultierenden Strings ist *strlen*(*dest*) + *strlen*(*src*).

Rückgabewert Zurückgeliefert wird ein Zeiger auf das erste Zeichen von *dest*.

Beispiel
```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char destination[25];
   char *blank = " ", *c = "C++", *turbo = "Turbo";

   strcpy(destination, turbo);
   strcat(destination, blank);
   strcat(destination, c);

   printf("%s\n", destination);
   return 0;
}
```

strchr, _fstrchr

Funktion

Sucht einen String nach dem ersten Auftreten eines bestimmten Zeichens ab.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strchr(const char *s, int c);
```

Far-Version:

```
char far * far _fstrchr(const char far *s, int c);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung

strchr sucht den über *s* angegebenen String nach dem Zeichen *c* ab, wobei die Suche mit dem *ersten* Zeichen von *s* beginnt. *strchr* findet die erste Fundstelle des Zeichens *c* im String *s*. Das abschließende Nullzeichen wird zum Stringinhalt gerechnet und als letztes verglichen – ein Aufruf wie

```
strchr(strs,0)
```

liefert also einen Zeiger auf das terminierende Nullzeichen von *strs*.

Rückgabewert

strchr liefert einen Zeiger auf die erste Fundstelle des Zeichens *c* im String *s* zurück bzw. den Wert NULL, wenn der String dieses Zeichen nicht enthält.

Siehe auch

strcspn, strrchr

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char string[15];
   char *ptr, c = 'r';

   strcpy(string, "This is a string");
   ptr = strchr(string, c);
   if (ptr)
      printf("The character %c is at position: %d\n",
             c, ptr-string);
   else
      printf("The character was not found\n");
   return 0;
}
```

strcmp, _fstrcmp

Funktion
Vergleicht zwei Strings miteinander.

Syntax
```
#include <string.h>
```
Near-Version:
```
int strcmp(const char *s1, const char *s2);
```
Far-Version:
```
int far _fstrcmp(const char far *s1,
                 const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung
strcmp führt mit den Inhalten von *s1* und *s2* einen **unsigned**-Vergleich durch. Beginnend mit dem ersten Zeichen wird zeichenweise verglichen bis das korrespondierende Zeichen ungleich ist oder das Ende der Strings erreicht wird.

Rückgabewert
strcmp liefert einen der folgenden Werte:

<0 wenn *s1* kleiner als *s2* ist,
$==0$ wenn *s1* gleich *s2* ist,
>0 wenn *s1* größer als *s2* ist.

Siehe auch
strcmpi, strcoll, stricmp, strncmp, strncmpi, strnicmp

Beispiel
```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
   int ptr;

   ptr = strcmp(buf2, buf1);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
   else
      printf("buffer 2 is less than buffer 1\n");

   ptr = strcmp(buf2, buf3);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 3\n");
   else
      printf("buffer 2 is less than buffer 3\n");

   return 0;
}
```

strcmpi

Funktion — Vergleicht zwei Strings ohne Unterscheidung zwischen Groß- und Kleinschreibung.

Syntax

```
#include <string.h>
int strcmpi(const char *s1, const char *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — *strcmpi* führt mit den Inhalten von *s1* und *s2* einen **unsigned** - Vergleich durch; zwischen Groß- und Kleinbuchstaben wird dabei nicht unterschieden.

Hinweis: strcmpi berücksichtigt die deutschen Umlaute nicht.

Die Funktionen *stricmp* und *strcmpi* haben dieselbe Wirkung; *strcmpi* ist ein in string.h implementiertes Makro, das Aufrufe von *strcmpi* in Aufrufe von *stricmp* übersetzt, *strcmpi* wurde aus Kompatibilitätsgründen zu anderen Compilern definiert. Um *strcmpi* zu verwenden, muß daher die Header-Datei string.h eingebunden werden.

Rückgabewert — *strcmpi* liefert einen der folgenden Werte:

<0 wenn *s1* kleiner als *s2* ist,
$==0$ wenn *s1* gleich *s2* ist,
>0 wenn *s1* größer als *s2* ist.

Siehe auch — *strcmp*, *strcoll*, *stricmp*, *strncmp*, *strncmpi*, *strnicmp*

Beispiel

```
/* strncmpi example */

#include <string.h>
#include <stdio.h>

int main(void)
{
   char *buf1 = "BBB", *buf2 = "bbb";
   int ptr;

   ptr = strcmpi(buf2, buf1);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

strcoll

Funktion Vergleicht zwei Strings.

Syntax
```
#include <string.h>
int strcoll(char *s1, char *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *strcoll* vergleicht den String, auf den *s1* zeigt, mit dem String, auf den *s2* zeigt. Dabei wird die mit *setlocale* gesetzte Vergleichsreihenfolge verwendet.

Rückgabewert *strcoll* liefert einen der folgenden Werte:

<0 wenn *s1* kleiner als *s2* ist,
$=0$ wenn *s1* gleich *s2* ist,
>0 wenn *s1* größer als *s2* ist.

Siehe auch *strcmp, strcmpi, stricmp, strncmp, strncmpi, strnicmp, strxfrm*

Beispiel
```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *two = "International";
   char *one = "Borland";
   int check;

   check = strcoll(one, two);
   if (check == 0)
      printf("The strings are equal\n");
   if (check <  0)
      printf("%s comes before %s\n", one, two);
   if (check >  0)
      printf("%s comes before %s\n", two, one);
   return 0;
}
```

strcpy, _fstrcpy

Funktion Kopiert einen String in einen anderen.

Syntax
```
#include <string.h>
```

Near-Version:
```
char *strcpy(char *dest, const char *src);
```

Far-Version:
```
char far *_fstrcpy(char far *sdest,
                   const char far *src);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strcpy* kopiert den Inhalt des über *src* angegebenen Strings in den durch *dest* angegebenen Speicherbereich. Das abschließende Nullzeichen von *src* wird als letztes Byte kopiert.

Rückgabewert *strcpy* liefert die Startadresse von *dest* zurück.

Siehe auch *stpcpy*

Beispiel
```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char string[10];
   char *str1 = "abcdefghi";

   strcpy(string, str1);
   printf("%s\n", string);
   return 0;
}
```

strcspn, _fstrcspn

Funktion Liefert die Länge des Teilstrings von *s1* zurück, der *keines* der Zeichen einer bestimmten Zeichenmenge enthält.

Syntax
```
#include <string.h>
```

Near-Version:
```
size_t strcspn(const char *s1, const char *s2);
```

Far-Version:
```
size_t far far _fstrcspn(const char far *s1,
                         const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Rückgabewert *strcspn* liefert die Länge des Teilstrings von *s1* zurück, der nur aus Zeichen besteht, die in *s2* nicht vorkommen.

Siehe auch *strchr, strrchr*

Beispiel

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>

int main(void)
{
   char *string1 = "1234567890";
   char *string2 = "747DC8";
   int length;

   length = strcspn(string1, string2);
   printf("Character where strings intersect is at "
          "position %d\n",
          length);

   return 0;
}
```

_strdate

Funktion Konvertiert das aktuelle Datum in einen String.

Syntax

```
#include <time.h>
char _strdate(char *buf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_strdate* konvertiert das aktuelle Datum in einen String und speichert diesen in dem durch *buf* angegebenen Puffer. *buf* muß mindestens 9 Zeichen lang sein.

Der String ist durch ein Nullzeichen abgeschlossen und hat das Format MM/DD/YY. MM, DD und YY sind Zahlen mit jeweils zwei Ziffern und geben Monat, Tag und Jahr an.

Rückgabewert Der Rückgabewert von *_strdate* ist *buf*, die Adresse des Datums-Strings.

Siehe auch *asctime*, *ctime*, *localtime*, *strftime*, *_strtime*, *time*

Beispiel

```
#include <time.h>
#include <stdio.h>
void main(void)
{
  char datebuf[9];
  char timebuf[9];

  _strdate(datebuf);
  _strtime(timebuf);
  printf("Date: %s  Time: %s\n",datebuf,timebuf);
}
```

strdup, _fstrdup

Funktion Kopiert einen String in einen neuen Speicherbereich.

Syntax
```
#include <string.h>
```
Near-Version:
```
char *strdup(const char *s);
```
Far-Version:
```
char far * far _fstrdup(const char far *s);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x			*Near-Version*
x		x			*Far-Version*

Beschreibung *strdup* bestimmt zuerst (mit *strlen*) die Länge des übergebenen Strings *s*, belegt danach (mit *malloc*) einen Speicherbereich entsprechender Größe (Länge von *s* +1) und kopiert den Inhalt von *s* in diesen neu belegten Speicherbereich. Der Programmierer muß den Speicher selbst wieder freigeben, wenn dieser nicht mehr benötigt wird.

Rückgabewert *strdup* liefert einen Zeiger auf den neu belegten Speicherbereich, der den duplizierten String enthält, zurück. Falls nicht genug Platz im Speicher ist (d.h. *malloc* einen Fehler liefert), dann ist das Funktionsergebnis von *strdup* NULL.

Siehe auch *free*

Beispiel
```
#include <stdio.h>
#include <string.h>
#include <alloc.h>

int main(void)
{
   char *dup_str, *string = "abcde";

   dup_str = strdup(string);
   printf("%s\n", dup_str);
   free(dup_str);

   return 0;
}
```

_strerror

Funktion Liefert Strings mit Fehlermeldungen.

Syntax
```
#include <string.h>
char *_strerror(const char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_strerror erlaubt das Generieren von eigenen Fehlermeldungen; die Funktion liefert einen Zeiger auf einen statischen String zurück, der die Fehlermeldung enthält.

Wenn für *s* der Zeigerwert NULL angegeben wird, dann führt *_strerror* die folgenden Schritte aus:

- die globale Variable *errno* wird gelesen – sie enthält die Nummer des zuletzt aufgetretenen Systemfehlers;
- der Wert von *errno* wird als Index innerhalb der Tabelle der Systemfehlermeldungen interpretiert, *_strerror* liefert einen Zeiger auf die entsprechende Meldung zurück.

Bezeichnet *s* einen existierenden String, liefert *_strerror* einen Zeiger auf einen String mit folgendem Inhalt zurück:

- zuerst kommt der über *s* übergebene String, der maximal 94 Zeichen enthalten darf;
- danach kommen ein Doppelpunkt und ein Leerzeichen;
- schließlich folgt der Text der durch *errno* ermittelten Systemfehlermeldung.

_strerror hat dieselbe Funktion wie *strerror* in der Version 1.0 von Turbo C.

Rückgabewert

_strerror liefert einen Zeiger auf einen statischen String mit der programmeigenen Fehlermeldung, der mit jedem Aufruf dieser Funktion erneut gesetzt wird.

Siehe auch

perror, strerror

Beispiel

```
#include <stdio.h>

int main(void)
{
   FILE *fp;

   /* open a file for writing */
   fp = fopen("TEST.$$$", "w");

   /* force an error condition by attempting to read */
   if (!fp) fgetc(fp);

   if (ferror(fp))
      /* display a custom error message */
      printf("%s", _strerror("Custom"));

   fclose(fp);
   return 0;
}
```

strerror

Funktion Liefert einen Zeiger auf einen String mit einer Fehlermeldung.

Syntax
```
#include <string.h>
char *strerror(int errnum);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *strerror* erwartet einen Integerwert als Parameter *errnum* und liefert einen Zeiger auf einen statischen String zurück, der die zum Wert von *errnum* gehörige Fehlermeldung enthält.

Rückgabewert *strerror* liefert einen Zeiger auf einen statischen Stringpuffer, der mit jedem Aufruf dieser Funktion erneut gesetzt wird.

Siehe auch *perror*, *_strerror*

Beispiel
```
#include <stdio.h>
#include <errno.h>

int main(void)
{
   char *buffer;
   buffer = strerror(errno);
   printf("Error: %s\n", buffer);
   return 0;
}
```

strftime

Funktion Formatiert die Uhrzeit für die Ausgabe.

Syntax
```
#include <time.h>
size_t strftime(char *s, size_t maxsize,
                const char *fmt, const struct tm *t);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *strftime* formatiert die Uhrzeit im Argument *t* und gibt sie in das Array aus, auf das das Argument *s* zeigt. Für die Formatierung gelten die Spezifikationen in *fmt*. Der Formatstring besteht aus Null oder mehr Anweisungen und normalen Zeichen. Eine Anweisung besteht aus dem Zeichen %, gefolgt von dem Zeichen für das einzusetzende Format. Alle normalen Zeichen werden unverändert kopiert. Es können maximal *maxsize* Zeichen in *s* plaziert werden.

Formatangabe	steht für
%%	Prozentzeichen %
%a	Abgekürzter Name des Wochentags
%A	Voller Name des Wochentags
%b	Abgekürzter Name des Monats
%B	Voller Name des Monats
%c	Datum und Zeit
%d	Tag des Monats mit zwei Ziffern (01 bis 31)
%H	Stunde mit zwei Ziffern (00 bis 23)
%I	Stunde mit zwei Ziffern (01 bis 12)
%j	Tag des Jahres mit drei Ziffern (001 bis 366)
%m	Monat als Dezimalzahl mit zwei Ziffern
%M	Minute mit zwei Ziffern (00 bis 59)
%p	AM oder PM
%S	Sekunde mit zwei Ziffern (00 bis 59)
%U	Wochennummer mit zwei Ziffern, wobei Sonntag der erste Tag der Woche ist (00 bis 53)
%w	Wochentag, wobei Sonntag 0 ist (0 bis 6)
%W	Wochennummer mit zwei Ziffern, wobei Montag der erste Tag der Woche ist (00 bis 53)
%x	Datum
%X	Uhrzeit
%y	Jahr mit zwei Ziffern ohne Jahrhundertangabe (00 bis 99)
%Y	Jahr mit Jahrhundertangabe
%Z	Name der Zeitzone oder keine Zeichen, wenn keine Zeitzone festgelegt ist

Rückgabewert *strftime* liefert die Anzahl der Zeichen zurück, die in *s* plaziert werden. Werden mehr Zeichen als in *maxsize* angegeben benötigt, ist der Rückgabewert 0.

Siehe auch *localtime*, *mktime*, *time*

Beispiel

```
#include <stdio.h>
#include <time.h>
#include <dos.h>

int main(void)
{
   struct tm *time_now;
   time_t secs_now;
   char str[80];

   tzset();
   time(&secs_now);
   time_now = localtime(&secs_now);
```

```
    strftime(str, 80, "It is %M minutes after %I "
            "o'clock (%Z)  %A, %B %d 19%y", time_now);
    printf("%s\n",str);
    return 0;
}
```

stricmp, _fstricmp

Funktion

Vergleicht zwei Strings ohne Unterscheidung zwischen Groß- und Kleinschreibung.

Syntax

```
#include <string.h>
```

Near-Version:

```
int stricmp(const char *s1, const char *s2);
```

Far-Version:

```
int far _fstricmp(const char far *s1,
                  const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

stricmp führt mit den Inhalten von *s1* und *s2* einen **unsigned**-Vergleich durch. Beginnend mit dem ersten Zeichen wird solange zeichenweise verglichen, bis das korrespondierende Zeichen differiert oder das Ende der Strings erreicht wird. Beim Vergleich werden Groß- und Kleinbuchstaben nicht unterschieden.

Die Routinen *stricmp* und *strcmpi* erfüllen dieselbe Funktion:

- *strcmpi* ist ein in string.h implementiertes Makro, das Aufrufe von *strcmpi* in Aufrufe von *stricmp* übersetzt;
- *strcmpi* wurde aus Kompatibilitätsgründen zu anderen Compilern definiert.

Um *strcmpi* zu verwenden, muß daher die Header-Datei string.h eingebunden werden.

Rückgabewert

stricmp liefert einen der folgenden Werte:

<0	wenn *s1* kleiner als *s2* ist;
=0	wenn *s1* gleich *s2* ist;
>0	wenn *s1* größer als *s2* ist.

Siehe auch

strcmp, *strcmpi*, *strcoll*, *strncmp*, *strncmpi*, *strnicmp*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
```

```
   char *buf1 = "BBB", *buf2 = "bbb";
   int ptr;

   ptr = stricmp(buf2, buf1);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

strlen,_fstrlen

Funktion — Berechnet die Länge eines Strings.

Syntax

```
#include <string.h>
```

Near-Version:

```
size_t strlen(const char *s);
```

Far-Version:

```
size_t *far _fstrlen(const char far *s);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung — *strlen* sucht nach dem Ende eines Strings *s* und liefert seine Länge zurück.

Rückgabewert — *strlen* liefert die Anzahl der Zeichen des Strings *s*; das abschließende Nullzeichen wird nicht mitgezählt.

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *string = "Borland International";

   printf("%d\n", strlen(string));
   return 0;
}
```

strlwr, _fstrlwr

Funktion Konvertiert alle Großbuchstaben in einem String in Kleinbuchstaben.

Syntax
```
#include <string.h>
```
Near-Version:
```
char *strlwr(char *s);
```
Far-Version:
```
char far * far _fstrlwr(char char far *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *strlwr* konvertiert sämtliche Großbuchstaben (A bis Z) des über *s* übergebenen Strings in Kleinbuchstaben (a bis z). Andere Zeichen werden nicht berücksichtigt.

Rückgabewert *strlwr* liefert einen Zeiger auf das erste Zeichen des String *s* zurück.

Siehe auch *strupr*

Beispiel
```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *string = "Borland International";

   printf("string prior to strlwr: %s\n", string);
   strlwr(string);
   printf("string after strlwr:    %s\n", string);
   return 0;
}
```

strncat, _fstrncat

Funktion Hängt eine bestimmte Anzahl von Zeichen aus einem String an einen anderen an.

Syntax
```
#include <string.h>
```
Near-Version:
```
char *strncat(char *dest, const char *src,
              size_t maxlen);
```
Far-Version:
```
char far *far _fstrncat(char far *dest,
                        const char far *src,
                        size_t maxlen);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strncat* hängt die ersten *maxlen* Zeichen von *src* an die bereits in *dest* vorhandenen Zeichen an. Die Operation wird beendet, wenn entweder *maxlen* Zeichen kopiert wurden oder das Ende von *src* erreicht ist. Als letztes schreibt *strncat* grundsätzlich ein Nullzeichen in *dest*.

Rückgabewert *strncat* liefert einen Zeiger auf das erste Zeichen von *dest* zurück.

Siehe auch *strcat*

Beispiel
```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char destination[25];
   char *source = " States";

   strcpy(destination, "United");
   strncat(destination, source, 7);
   printf("%s\n", destination);
   return 0;
}
```

strncmp, _fstrncmp

Funktion Vergleicht eine bestimmte Anzahl von Zeichen zweier Strings miteinander.

Syntax
```
#include <string.h>
```
Near-Version:
```
int strncmp(const char *s1, const char *s2,
            size_t maxlen);
```
Far-Version:
```
int far _fstrncmp(const char far *s1,
                  const char far *s2,
                  size_t maxlen);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strncmp* führt den gleichen **unsigned**-Vergleich wie *strcmp* durch, vergleicht aber maximal die ersten *maxlen* Zeichen von

s1 und *s2* miteinander. Beginnend mit dem ersten Zeichen wird zeichenweise verglichen, bis das korrespondierende Zeichen differiert oder *maxlen* Zeichen untersucht worden sind.

Rückgabewert *strncmp* liefert einen der folgenden Integerwerte zurück:

<0 wenn *s1* kleiner als *s2* ist;
=0 wenn *s1* gleich *s2* ist;
>0 wenn *s1* größer als *s2* ist.

Siehe auch *strcmp*, *strcoll*, *stricmp*, *strncmpi*, *strnicmp*

Beispiel

```
#include <string.h>
#include <stdio.h>

int  main(void)

{
   char *buf1 = "aaabbb", *buf2 = "bbbccc",
                *buf3 = "ccc";
   int ptr;

   ptr = strncmp(buf2,buf1,3);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
   else
      printf("buffer 2 is less than buffer 1\n");

   ptr = strncmp(buf2,buf3,3);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 3\n");
   else
      printf("buffer 2 is less than buffer 3\n");

   return(0);
}
```

strncmpi

Funktion Vergleicht eine bestimmte Anzahl von Zeichen zweier Strings miteinander ohne Unterscheidung zwischen Groß- und Kleinschreibung.

Syntax

```
#include <string.h>
int strncmpi(const char *s1, const char *s2,
             size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *strncmpi* vergleicht (in einem **signed**-Vergleich) maximal *n* Zeichen von *s1* und *s2* miteinander. Es wird kein Unterschied

zwischen Groß- und Kleinschreibung gemacht. Beginnend mit dem ersten Zeichen wird zeichenweise solange verglichen, bis das korrespondierende Zeichen differiert oder *n* Zeichen untersucht worden sind.

Die Routinen *strnicmp* und *strncmpi* erfüllen dieselbe Funktion; *strncmpi* ist ein in string.h implementiertes Makro, das Aufrufe von *strncmpi* in Aufrufe von *strnicmp* übersetzt. *strncmpi* wurde aus Kompatibilitätsgründen zu anderen Compilern definiert. Um *strncmpi* zu verwenden, muß daher die Include-Datei string.h eingebunden werden.

Rückgabewert *strncmpi* liefert einen der folgenden Integerwerte zurück:

<0 wenn *s1* kleiner als *s2* ist;
=0 wenn *s1* gleich *s2* ist;
>0 wenn *s1* größer als *s2* ist.

Siehe auch *strnicmp*

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char *buf1 = "BBBccc", *buf2 = "bbbccc";
   int ptr;

   ptr = strncmpi(buf2,buf1,3);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

strncpy, _fstrncpy

Funktion Kopiert eine bestimmte Anzahl von Bytes von einem String in einen anderen, wobei gegebenenfalls verkürzt oder aufgefüllt wird.

Syntax

```
#include <stdio.h>
```

Near-Version:

```
char *strncpy(char *dest, const char *src,
              size_t maxlen);
```

Far-Version:
```
char far *far _fstrncpy(char far *dest,
                        const char far *src,
                        size_t maxlen);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strncpy* kopiert *maxlen* Zeichen des über *src* angegebenen Strings in den durch *dest* angegebenen Speicherbereich. *strncpy* kopiert immer genau *maxlen* Zeichen – wenn *src* weniger als *maxlen* Zeichen enthält, wird eine entsprechende Anzahl von Nullzeichen in *dest* angehängt; wenn *src* mehr als *maxlen* Zeichen umfaßt, bleibt der nach *dest* kopierte String ohne ein abschließendes Nullzeichen.

Rückgabewert *strncpy* liefert in allen Fällen einen Zeiger auf das erste Byte von *dest* zurück.

Beispiel
```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char string[10];
   char *str1 = "abcdefghi";

   strncpy(string, str1, 3);
   string[3] = '\0';
   printf("%s\n", string);
   return 0;
}
```

strnicmp, _fstrnicmp

Funktion Vergleicht eine bestimmte Anzahl von Zeichen zweier Strings ohne Unterscheidung zwischen Groß- und Kleinschreibung.

Syntax
```
#include <string.h>
```

Near-Version:
```
int strnicmp(const char *s1, const char *s2,
             size_t maxlen);
```

Far-Version:
```
int far _fstrnicmp(const char far *s1,
                   const char far *s2, size_t maxlen);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *strnicmp* vergleicht (in einem **signed**-Vergleich) maximal *maxlen* Zeichen von *s1* und *s2* miteinander. Es wird kein Unterschied zwischen Groß- und Kleinschreibung gemacht. Beginnend mit dem ersten Zeichen wird zeichenweise verglichen bis das korrespondierende Zeichen differiert oder das Ende des Strings erreicht ist.

Rückgabewert *strnicmp* liefert einen der folgenden Werte zurück:

<0 wenn *s1* kleiner als *s2* ist;
$=0$ wenn *s1* gleich *s2* ist;
>0 wenn *s1* größer als *s2* ist.

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char *buf1 = "BBBccc", *buf2 = "bbbccc";
   int ptr;

   ptr = strnicmp(buf2, buf1, 3);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

strnset, _fstrnset

Funktion Füllt eine bestimmte Anzahl von Zeichen eines Strings mit einem bestimmten Zeichen.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strnset(char *s, int ch, size_t n);
```

Far-Version:

```
char far *far _fstrnset(char far *s, int ch, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *strnset* überschreibt die ersten *n* Zeichen des über *s* übergebenen Strings mit dem Zeichen *ch*. Wenn *n* > *strlen*(*s*) ist, dann wird *n* durch *strlen*(*s*) ersetzt. *strnset* beendet die Operation, wenn *n* Zeichen überschrieben wurden oder wenn das abschließende Nullzeichen erreicht wird.

Rückgabewert *strnset* liefert einen Zeiger auf das erste Zeichen von *s* zurück.

Beispiel
```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *string = "abcdefghijklmnopqrstuvwxyz";
   char letter = 'x';

   printf("string before strnset: %s\n", string);
   strnset(string, letter, 13);
   printf("string after  strnset: %s\n", string);

   return 0;
}
```

strpbrk, _fstrpbrk

Funktion Sucht einen String nach dem ersten Vorkommen bestimmter Zeichen ab.

Syntax
```
#include <string.h>
```

Near-Version:
```
char *strpbrk(const char *s1, const char *s2);
```

Far-Version:
```
char far *far _fstrpbrk(const char far *s1,
                        const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strpbrk* sucht den über *s1* angegebenen String nach dem ersten Vorkommen eines in *s2* enthaltenen Zeichens ab.

Rückgabewert Die Funktion liefert einen Zeiger auf die erste Fundstelle bzw. den Wert NULL zurück, wenn keines der durch *s2* festgelegten Zeichen in *s1* enthalten ist.

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *string1 = "abcdefghijklmnopqrstuvwxyz";
   char *string2 = "onm";
   char *ptr;

   ptr = strpbrk(string1, string2);

   if (ptr)
      printf("strpbrk found first character: %c\n",
             *ptr);
   else
      printf("strpbrk didn't find character in set\n");

   return 0;
}
```

strrchr, _fstrrchr

Funktion

Sucht einen String nach dem letzten Vorkommen eines bestimmten Zeichens ab.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strrchr(const char *s, int c);
```

Far-Version:

```
char far *far _fstrrchr(const char far *s, int c);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung

strrchr sucht den über *s* angegebenen String nach dem Zeichen *c* ab, wobei die Suche mit dem *letzten* Zeichen von *s* beginnt. Das abschließende Nullzeichen wird zum Stringinhalt gerechnet und als erstes verglichen.

Rückgabewert

strrchr liefert einen Zeiger auf das letzte Vorkommen des Zeichens *c* in *s* zurück bzw. den Wert NULL, wenn der String dieses Zeichen nicht enthält.

Siehe auch

strcspn, *strchr*

Beispiel

```
#include <string.h>
#include <stdio.h>
```

```
int main(void)
{
   char string[15];
   char *ptr, c = 'r';

   strcpy(string, "This is a string");
   ptr = strrchr(string, c);
   if (ptr)
      printf("The character %c is at position: %d\n",
             c, ptr-string);
   else
      printf("The character was not found\n");
   return 0;
}
```

strrev, _fstrrev

Funktion

Dreht die Reihenfolge der Zeichen eines Strings um.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strrev(char *s);
```

Far-Version:

```
char far *far _fstrrev(char far *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

strrev spiegelt die Reihenfolge sämtlicher Zeichen des über *s* übergebenen Strings (aus »Meier« wird »reieM«). Das abschließende Nullzeichen bleibt unverändert.

Rückgabewert

strrev liefert einen Zeiger auf den gespiegelten String zurück.

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char *forward = "string";

   printf("Before strrev(): %s\n", forward);
   strrev(forward);
   printf("After strrev():  %s\n", forward);
   return 0;
}
```

strset, _fstrset

Funktion

Überschreibt sämtliche Zeichen eines Strings mit einem bestimmten Zeichen.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strset(char *s, int ch);
```

Far-Version:

```
char far *far _fstrset(char far *s, int ch);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

strset überschreibt sämtliche Zeichen des über *s* übergebenen Strings mit dem Zeichen *ch* und bricht erst beim abschließenden Nullzeichen ab.

Rückgabewert

strset liefert einen Zeiger auf das erste Zeichen von *s* zurück.

Siehe auch

setmem

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char string[10] = "123456789";
   char symbol = 'c';

   printf("Before strset(): %s\n", string);
   strset(string, symbol);
   printf("After strset():  %s\n", string);
   return 0;
}
```

strspn, _fstrspn

Funktion

Liefert die Länge des Teilstrings von *s1* zurück, der ausschließlich aus in *s2* angegebenen Zeichen besteht.

Syntax

```
#include <string.h>
```

Near-Version:

```
size_t strspn(const char *s1, const char *s2);
```

Far-Version:

```
size_t far _fstrspn(const char far *s1,
                    const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung Beginnend mit dem ersten Zeichen durchsucht *strspn s1* nach dem Vorkommen der Zeichen in *s2*. Die Funktion stoppt beim Vorkommen des ersten nicht übereinstimmenden Zeichens.

Rückgabewert *strspn* liefert die Länge des Teilstrings von *s1*, der ausschließlich aus Zeichen besteht, die in *s2* enthalten sind.

Beispiel

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>

int main(void)
{
   char *string1 = "1234567890";
   char *string2 = "123DC8";
   int length;

   length = strspn(string1, string2);
   printf("Character where strings differ is "
          "at position %d\n", length);
   return 0;
}
```

strstr, _fstrstr

Funktion Sucht einen String nach dem Vorkommen eines bestimmten Teilstrings ab.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strstr(const char *s1, const char *s2);
```

Far-Version:

```
char far *far _fstrstr(const char far *s1,
                       const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung *strstr* sucht den über *s1* angegebenen String nach dem ersten Vorkommen des Teilstrings *s2* ab.

Rückgabewert *strstr* liefert einen Zeiger auf den Beginn von *s2* in *s1* zurück bzw. den Zeigerwert NULL, wenn *s2* in *s1* überhaupt nicht vorkommt.

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *str1 = "Borland International",
                *str2 = "nation", *ptr;

   ptr = strstr(str1, str2);
   printf("The substring is: %s\n", ptr);
   return 0;
}
```

_strtime

Funktion Konvertiert die aktuelle Uhrzeit in einen String.

Syntax

```
#include <time.h>
char *_strtime(char *buf);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *_strtime* konvertiert die aktuelle Uhrzeit in einen String und speichert diesen in dem durch *buf* angegebenen Puffer. *buf* muß mindestens 9 Zeichen lang sein.

Der String ist durch ein Nullzeichen abgeschlossen und hat das Format HH:MM:SS. HH, MM und SS sind Zahlen mit jeweils zwei Ziffern und geben Stunde, Minute und Sekunde an. Der String ist nullterminiert.

Rückgabewert Der Rückgabewert von *_strtime* ist *buf*, die Adresse des Uhrzeit-Strings.

Siehe auch *asctime*, *ctime*, *localtime*, *strftime*, *_strdate*, *time*

Beispiel

```
#include <time.h>
#include <stdio.h>
void main(void)
{
  char datebuf[9];
  char timebuf[9];

  _strdate(datebuf);
  _strtime(timebuf);
  printf("Date: %s  Time: %s\n",datebuf,timebuf);
}
```

strtod, _strtold

Funktion

Konvertiert einen String in einen **double** bzw. **long double** -Wert.

Syntax

```
#include <stdlib.h>
double strtod(const char *s, char **endptr);
long double _strtold(const char *s, char **endptr);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*strtod*
x		x			*_strtold*

Beschreibung

strtod interpretiert den über *s* übergebenen String als **double** Der Stringinhalt muß das folgende Format haben:

```
[ws] [sn] [ddd] [.] [ddd] [fmt [sn] ddd]
```

[*ws*] steht hier für ein optionales »Whitespace«, [*sn*] für ein optionales Vorzeichen und [*ddd*] für eine oder mehrere optionale Dezimalziffern, [*fmt*] für einen optionalen Exponenten (*e* oder *E*). Wenn ein Exponent angegeben wird, dann kann dieser ein Vorzeichen enthalten. Beispiele für gültige Angaben sind:

```
+1234.1981e-1
502.85E2
+ 2010.952
```

strtod erkennt auch +INF und -INF für plus/minus unendlich und +NAN und -NAN für eine »Nichtzahl« (Not-a-Number).

strtod bricht die Analyse beim ersten Zeichen ab, das nicht mehr als Teil eines **double**Werts interpretiert werden kann.

Solange der Parameter *endptr* nicht NULL ist, wird **endptr* von *strtod* auf das Zeichen innerhalb von *s* gesetzt, durch das die Analyse abgebrochen wurde (**endptr* = *&stopper*). *endptr* ist nützlich für die Fehlersuche.

_strtold ist die **long double** Version. Sie konvertiert einen String in einen Wert vom Typ **long double** .

Rückgabewert

Diese Funktionen liefern das Ergebnis der Interpretation von *s* als **double**(*strtod*) oder als **long double** (*_strtold*) zurück. Im Falle eines Überlaufs wird plus oder minus HUGE_VAL bzw. _LHUGE_VAL zurückgeliefert.

Siehe auch

atof

Beispiel

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
```

```
{
   char input[80], *endptr;
   double value;

   printf("Enter a floating point number:");
   gets(input);
   value = strtod(input, &endptr);
   printf("The string is %s the number is %lf\n",
          input, value);
   return 0;
}
```

strtok, _fstrok

Funktion

Sucht einen String nach einer von mehreren Zeichenfolgen ab, die in einem zweiten String definiert sind.

Syntax

```
#include <string.h>
```

Near-Version:

```
char *strtok(char *s1, const char *s2);
```

Far-Version:

```
char far *far _fstrtok(char far *s1,
                       const char far *s2);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x	x	x	x		*Near-Version*
x		x			*Far-Version*

Beschreibung

strtok betrachtet den über *s1* angegebenen String als Folge von Null oder mehr Zeichenfolgen, die voneinander durch eines oder mehrere der in *s2* definierten Zeichen getrennt sind.

Der erste Aufruf von *strtok* liefert einen Zeiger auf den Beginn der ersten gefundenen Zeichenfolge in *s1* zurück und setzt ein Nullzeichen unmittelbar hinter das Ende der Zeichenfolge. Folgende Aufrufe von *strtok* mit dem Wert NULL anstelle des Arguments *s1* liefern die restlichen Zeichenfolgen in *s1*. *s2* kann bei jedem Aufruf von *strtok* beliebig geändert werden.

Rückgabewert

strtok liefert einen Zeiger auf den Beginn der ersten gefundenen Zeichenfolge in *s1* zurück. Wenn keine weitere Zeichenfolge gefunden wird, ist der Rückgabewert NULL.

Beispiel

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   char input[16] = "abc,d";
```

```
    char *p;

    /* strtok places a NULL terminator
    in front of the token, if found */
    p = strtok(input, ",");
    if (p)   printf("%s\n", p);

    /* A second call to strtok using a NULL
    as the first parameter returns a pointer
    to the character following the token  */
    p = strtok(NULL, ",");
    if (p)   printf("%s\n", p);
    return 0;
}
```

strtol

Funktion Konvertiert einen String in einen **long**-Wert.

Syntax
```
#include <stdlib.h>
long strtol(const char *s, char **endptr, int radix)
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *strtol* konvertiert den über *s* übergebenen String in einen **long**-Integerwert. Die in *s* enthaltene Zeichenfolge muß das folgende Format haben (vgl. *atol*):

```
[ws] [sn] [0] [x|X] [ddd]
```

[*ws*] steht für ein optionales »Whitespace«, [*sn*] für ein optionales Vorzeichen (+ oder -) und [*0*] für eine optionale Null. [*x*] ist ein optionales x oder X, [*ddd*] eine oder mehrere optionale Ziffern.

Liegt *radix* zwischen 2 und 36, so wird *s* im Zahlensystem zur Basis *radix* interpretiert. Ist *radix* 0, so legen die ersten Zeichen von *s* die Basis fest:

Erstes Zeichen	Zweites Zeichen	Interpretation
0	1 bis 7	Oktalzahl
0	x oder X	Hexadezimalzahl
1 bis 9		Dezimalzahl

Ist *radix* gleich 1, kleiner als 0 oder größer als 36, so wird der Parameter als ungültiger Wert interpretiert.

Bei Ausgabe ungültiger Werte für *radix* liefert die Funktion

Null zurück und setzt den nächsten Stringzeiger **endptr* auf den Stringanfang.

Falls der Wert in *s* als Oktalzahl interpretiert werden soll, werden nur die Zeichen zwischen 0 und 7 erkannt.

Soll der Wert in *s* als Dezimalzahl interpretiert werden, werden nur die Zeichen 0 bis 9 erkannt.

Falls der Wert in *s* als Zahl eines Zahlensystems zu einer anderen Basis interpretiert werden soll, werden nur die Zeichen, die zur Repräsentation von Zahlen dieses Zahlensystems verwendet werden, erkannt. (Beispielsweise nur die Zeichen 0 bis 4, wenn *radix* gleich 5 ist, ist *radix* gleich 20, werden nur die Zeichen 0 bis 9 und A bis J erkannt.)

strtol bricht die Analyse beim ersten Zeichen ab, das nicht mehr als dem Wert zugehörig interpretiert werden kann. Wenn für den Parameter *endptr* die Adresse eines Zeigers angegeben wurde (und nicht der Wert NULL), dann wird **endptr* auf die Adresse des ersten nicht analysierbaren Zeichens in *s* gesetzt (**endptr* = *&stopper*).

Rückgabewert *strtol* liefert das Ergebnis der Interpretation als **long**, bei einem Fehler 0 zurück.

Siehe auch *atoi, atol, strtoul*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   char *string = "87654321", *endptr;
   long lnumber;

   /* strtol converts string to long integer  */
   lnumber = strtol(string, &endptr, 10);
   printf("string = %   s  long = %ld\n", string,
          lnumber);

   return 0;
}
```

strtoul

Funktion Konvertiert einen String in einen **unsigned long** Wert des durch *radix* gegebenen Zahlensystems.

Syntax

```
#include <stdlib.h>
unsigned long strtoul(const char *s, char **endptr,
                      int radix);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *strtoul* arbeitet genauso wie *strtol*, interpretiert die als *s* übergebene Zeichenfolge aber als **unsigned long** (siehe die Beschreibung von *strtol*).

Rückgabewert *strtoul* liefert das Ergebnis der Interpretation als **unsigned long**, bei einem Fehler 0 zurück.

Siehe auch *atol, strtol*

Beispiel
```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   char *string = "87654321", *endptr;
   unsigned long lnumber;

   lnumber = strtoul(string, &endptr, 10);
   printf("string = %  s  long = % lu\n",
   string, lnumber);

   return 0;
}
```

strupr, _fstrupr

Funktion Konvertiert alle Kleinbuchstaben eines Strings in Großbuchstaben.

Syntax
```
#include <string.h>
```
Near-Version:
```
char *strupr(char *s);
```
Far-Version:
```
char far *far _fstrupr(char far *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung *strupr* verändert sämtliche Kleinbuchstaben (a bis z) des mit *s* übergebenen Strings in Großbuchstaben (A bis Z). Andere Zeichen werden nicht berücksichtigt.

Rückgabewert *strupr* liefert einen Zeiger auf das erste Zeichen des Strings *s* zurück.

Siehe auch *strlwr*

Beispiel

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *string = "abcdefghijklmnopqrstuvwxyz", *ptr;

   /* converts string to upper case characters */
   ptr = strupr(string);
   printf("%s\n", ptr);
   return 0;
}
```

strxfrm

Funktion Kopiert einen Teilstring in einen anderen String.

Syntax

```
#include <string.h>
size_t strxfrm(char *s1, char *s2, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x	x	

Beschreibung *strxfrm* kopiert den String, auf den *s2* zeigt, in den String *s1*, wobei nicht mehr als *n* Zeichen kopiert werden.

Rückgabewert *strxfrm* liefert die Anzahl der kopierten Zeichen zurück.

Siehe auch *strcoll, strncpy*

Beispiel

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>

int main(void)
{
   char *target;
   char *source = "Frank Borland";
   int length;

   /* allocate space for the target string */
   target = (char *) calloc(80, sizeof(char));

   /* copy the source over to the target
      and get the length */
   length = strxfrm(target, source, 80);

   /* print out the results */
   printf("%s has the length %d\n", target, length);
   return 0;
}
```

swab

Funktion Vertauscht Bytepositionen.

Syntax
```
#include <stdlib.h>
void swab(char *from, char *to, int nbytes);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *swab* liest *nbytes* Bytes von dem durch *from* bezeichneten Speicherbereich, tauscht jeweils die »Worthälften« miteinander aus (das erste Byte wird mit dem zweiten vertauscht, das dritte mit dem vierten usw.), und schreibt diese vertauschten Bytes in den durch *to* bezeichneten Speicherbereich.

nbytes sollte einen geradzahligen Wert haben.

Sinnvoll ist *swab* hauptsächlich dann, wenn Daten zwischen Maschinen mit Prozessoren, welche eine unterschiedliche Byte-Reihenfolge haben, ausgetauscht werden sollen.

Rückgabewert Keiner.

Beispiel
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char source[15] = "rFna koBlrna d";
char target[15];

int main(void)
{
   swab(source, target, strlen(source));
   printf("This is target: %s\n", target);
   return 0;
}
```

system

Funktion Führt einen DOS-Befehl aus.

Syntax
```
#include <stdlib.h>
int system(const char *command);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung *system* lädt das Programm COMMAND.COM in den Speicher und übergibt den über *command* angegebenen String als Kommandozeile. Die in diesem String enthaltenen Befehle (DOS-Befehle, Stapeldatei- oder Programmaufrufe) werden von

COMMAND.COM in derselben Weise ausgeführt, als ob sie auf der DOS-Kommandoebene über die Tastatur eingegeben worden wären.

Aufgerufene Programme müssen sich im aktuellen Verzeichnis bzw. in einem im PATH-String aufgeführten Suchweg befinden.

Für die Suche nach COMMAND.COM wird der *Environment*-Eintrag COMSPEC verwendet – COMMAND.COM muß sich also nicht unbedingt innerhalb des aktuellen Verzeichnisses befinden.

Rückgabewert *system* liefert einen Wert ungleich Null, falls *command* ein Nullzeiger ist und der Befehlsprozessor zur Verfügung steht. Ist *command* kein Nullzeiger, so liefert *system* Null, falls der Befehlsprozessor erfolgreich gestartet wurde. Bei einem Fehler wird der Wert -1 zurückgeliefert und *errno* auf einen der Werte ENOENT, ENOMEM, E2BIG oder ENOEXEC gesetzt.

Siehe auch *exec..., _fpreset, searchpath, spawn...*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   printf("About to spawn command.com "
          "and run a DOS command\n");
   system("dir");
   return 0;
}
```

tan, tanl

Funktion Liefert den Tangens des Arguments.

Syntax

reelle Versionen:

```
#include <math.h>
double tan(double x);
long double tanl(long double x);
```

komplexe Version:

```
#include <complex.h>
complex tan(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*tanl*
x	x	x	x		*Reelles tan*
x		x		x	*Komplexes tan*

Beschreibung *tan* erwartet einen Winkel in der Einheit *rad* als Argument und liefert den dazugehörigen Tangens ($sin(x)/cos(x)$) zurück.

tanl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Die Fehlerbehandlung für *tan* und *tanl* kann mit *matherr* bzw. *_matherrl* modifiziert werden.

Die komplexe Version von *tan* ist definiert als:

$$tan(z) = sin(z) / cos(z)$$

Rückgabewert *tan* und *tanl* liefern den Tangens von *x* zurück.

Siehe auch *acos*, *asin*, *atan*, *atan2*, *complex*, *cos*, *sin*

Beispiel

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x;

   x = 0.5;
   result = tan(x);
   printf("The tan of %lf is %lf\n", x, result);
   return 0;
}
```

tanh, tanhl

Funktion Liefert den Tangens hyperbolicus des Arguments.

Syntax

reelle Versionen:

```
#include <math.h>
double tanh(double x);
long double tanhl(long double x);
```

komplexe Version:

```
#include <complex.h>
complex tanh(complex x);
```

DOS	UNIX	Windows	ANSI C	nur C++	
x		x			*tanhl*
x	x	x	x		*Reelles tanh*
x		x		x	*Komplexes tanh*

Beschreibung *tanh* berechnet den Tangens hyperbolicus ($sinh(x) / cosh(x)$).

tanhl ist die **long double** Version. Sie erwartet ein Argument vom Typ **long double** und liefert ein Ergebnis vom Typ **long double** zurück.

Die Fehlerbehandlung für *tanh* und *tanhl* kann über die Routine *matherr* bzw. *_matherrl* modifiziert werden.

Die komplexe Version von *tanh* ist definiert als:

$$tanh(z) = sinh(z) / cosh(z)$$

Rückgabewert *tanh* und *tanhl* liefern den Tangens hyperbolicus von *x*.

Siehe auch *complex*, *cos*, *cosh*, *sin*, *sinh*, *tan*

Beispiel

```
#include <stdio.h>
#include <math.h>
```

```
int main(void)
{
   double result, x;

   x = 0.5;
   result = tanh(x);
   printf("The hyperbolic tangent of %lf is %lf\n",
          x, result);
   return 0;
}
```

tell

Funktion

Liefert die aktuelle Position eines Dateizeigers zurück.

Syntax

```
#include <io.h>
long tell(int handle);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

tell liefert die aktuelle Position des Dateizeigers innerhalb der durch *handle* bezeichneten Datei zurück (relativ zum Dateianfang, Zählung in Bytes).

Rückgabewert

tell liefert die aktuelle Position des Dateizeigers innerhalb der angegebenen Datei als **long** zurück – falls sich bei dieser Operation ein Fehler ereignen sollte, ist der Rückgabewert *-1L* und *errno* bekommt den Wert

EBADF Ungültiges Handle.

Siehe auch

fgetpos, fseek, ftell, lseek

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = open("TEST.$$$",
                      O_CREAT | O_TEXT | O_APPEND)) == -1)
   {
      perror("Error:");
      return 1;
   }
   write(handle, msg, strlen(msg));
```

```
    printf("The file pointer is at byte %ld\n",
           tell(handle));
    close(handle);
    return 0;
}
```

tempnam

Funktion

Erzeugt einen eindeutigen Dateinamen im angegebenen Verzeichnis.

Syntax

```
#include <stdio.h>
char *tempnam(char *dir, char *prefix);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung

tempnam versucht, in den folgenden Verzeichnissen (in der angegebenen Reihenfolge) einen eindeutigen Dateinamen anzulegen:

- In dem durch den *Environment*-Eintrag TMP angegebenen Verzeichnis.
- In dem Verzeichnis, das mit *dir* an *tempnam* übergeben wird.
- Im Verzeichnis, das in der *P_tmpdir*-Definition in stdio.h angegeben ist. Wenn Sie dieses Verzeichnis in stdio.h ändern, wird die neue Definition nicht von *tempnam* verwendet.
- Im aktuellen Arbeitsverzeichnis.

Wenn eines der aufgeführten Verzeichnisse NULL, nicht definiert oder nicht-existent ist, wird es ignoriert.

Der Parameter *prefix* enthält den ersten Teil des Dateinamens. Dieser darf nicht länger als fünf Zeichen sein und keinen Punkt (.) enthalten. Durch die Verknüpfung des Verzeichnisnamens mit dem Parameter *prefix* und 6 zusätzlichen Zeichen wird ein eindeutiger Dateiname erzeugt.

Der Speicherplatz für den resultierenden Dateinamen wird mit *malloc* reserviert. Wenn der Dateiname nicht mehr benötigt wird, sollte die aufrufende Funktion den Speicherplatz durch einen Aufruf von *free* wieder freigeben. Die Datei selbst wird durch den Aufruf von *tempnam* nicht erzeugt – die Funktion überprüft lediglich, ob der Name bereits existiert.

Hinweis: Im Gegensatz zur Funktion *tmpfile* löscht *tempnam* den Dateinamen nicht. Wenn Sie für den mit *tempnam* erzeugten Namen eine temporäre Datei erstellen, liegt es deshalb in Ihrer Verantwortung, den Dateinamen vor dem Ende eines Programms wieder zu löschen (z.B. mit *remove*).

Rückgabewert Bei fehlerfreier Ausführung liefert *tempnam* einen Zeiger auf den eindeutigen Namen einer temporären Datei. Dieser Zeiger kann später (wenn der Dateiname nicht mehr benötigt wird) an *free* übergeben werden. Wenn *tempnam* keinen eindeutigen Dateinamen zurückliefern kann, ist das Funktionsergebnis NULL.

Siehe auch *mktemp*, *tmpfile*, *tmpnam*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
  FILE *stream;
  int i;
  char *name;

  for (i = 1; i <= 10; i++) {
    if ((name = tempnam("\\tmp","wow")) == NULL)
      perror("tempnam couldn't create name");
    else {
      printf("Creating %s\n",name);
      if ((stream = fopen(name,"wb")) == NULL)
        perror("Could not open temporary file\n");
      else
        fclose(stream);
    }
    free(name);
  }
  printf("Warning: temp files not deleted.\n");
}
```

textattr

Funktion Setzt das Attribut für folgende Textausgaben.

Syntax

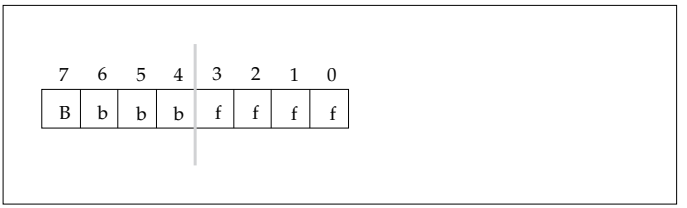
```
#include <conio.h>
void textattr(int newattr);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *textattr* setzt sowohl die Vordergrund- als auch die Hintergrundfarbe für folgende Textausgaben (d.h. vereinigt sozusagen die Funktionen *textcolor* und *textbackground*). Diese Funktion läßt bereits auf dem Bildschirm befindliche Zeichen unverändert. Sie verändert nur die Zeichen, die im Textmodus

von Funktionen (wie *cprintf*) direkt auf den Bildschirm ausgegeben werden, nachdem die Funktion aufgerufen wurde.

Der Parameter *newattr* hat die folgende Struktur:



Die vier mit *ffff* bezeichneten Bits stehen für die Vordergrundfarbe (Werte von 0 bis 15), die mit *bbb* bezeichneten Bits für die Hintergrundfarbe (Werte von 0 bis 7); *B* steht für das Attribut »Blinken« und ist in conio.h als Konstante BLINK mit dem Wert 128 definiert.

Wenn die in conio.h definierten Farbkonstanten zur Zuordnung von Textattributen über *textattr* verwendet werden sollen, müssen folgende Beschränkungen beachtet werden:

- Sie können nur eine der acht ersten Farben für den Hintergrund wählen.
- Die Hintergrundfarbe muß um 4 Bitpositionen nach links geschoben werden.

Für Vorder- und Hintergrundfarben sind die folgenden symbolischen Konstanten in conio.h definiert:

Konstante	Wert	Beschreibung	Vorder- o. Hintergrund?
BLACK	0	schwarz	Für beide
BLUE	1	blau	Für beide
GREEN	2	grün	Für beide
CYAN	3	türkis	Für beide
RED	4	rot	Für beide
MAGENTA	5	violett	Für beide
BROWN	6	braun	Für beide
LIGHTGRAY	7	hellgrau	Für beide
DARKGRAY	8	dunkelgrau	Nur für Vordergrund
LIGHTBLUE	9	hellblau	Nur für Vordergrund
LIGHTGREEN	10	hellgrün	Nur für Vordergrund
LIGHTCYAN	11	helltürkis	Nur für Vordergrund
LIGHTRED	12	hellrot	Nur für Vordergrund
LIGHTMAGENTA	13	hellviolett	Nur für Vordergrund
YELLOW	14	gelb	Nur für Vordergrund
WHITE	15	weiß	Nur für Vordergrund
BLINK	128	»blinkend«	Nur für Vordergrund

Rückgabewert Keiner.

Siehe auch *gettextinfo, highvideo, lowvideo, normvideo, textbackground, textcolor*

Beispiel

```
#include <conio.h>

int main(void)
{
   int i;

   clrscr();
   for (i=0; i<9; i++)
   {
       textattr(i + ((i+1) << 4));
       cprintf("This is a test\r\n");
   }

   return 0;
}
```

textbackground

Funktion Legt die Hintergrundfarbe für nachfolgende Textausgaben fest.

Syntax

```
#include <conio.h>
void textbackground(int newcolor);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *textbackground* legt die Hintergrundfarbe für folgende Zeichenausgaben fest. Der Parameter *newcolor* kann auf einen Integerwert zwischen 0 bis 7 oder auf eine der in conio.h definierten Konstanten gesetzt werden. Falls Sie symbolische Konstanten verwenden, muß die Include-Datei conio.h eingebunden werden.

Wenn *textbackground* aufgerufen wurde, verwenden alle folgenden Aufrufe von Funktionen mit direkter Bildschirmausgabe (wie z.B. *cprintf*) den Parameter *newcolor*. Bereits auf dem Bildschirm befindliche Zeichen bleiben unverändert.

Die Werte für die Hintergrundfarbe entsprechen folgenden in conio.h definierten symbolischen Konstanten:

Name	Wert	Beschreibung
BLACK	0	schwarz
BLUE	1	blau
GREEN	2	grün
CYAN	3	türkis
RED	4	rot
MAGENTA	5	violett

Name	Wert	Beschreibung
BROWN	6	braun
LIGHTGRAY	7	hellgrau

Rückgabewert Keiner.

Siehe auch *gettextinfo, textattr, textcolor*

Beispiel

```
#include <conio.h>

int main(void)
{
   int i, j;

   clrscr();
   for (i=0; i<9; i++)
   {
       for (j=0; j<80; j++)
          cprintf("C");
       cprintf("\r\n");
       textcolor(i+1);
       textbackground(i);
   }

   return 0;
}
```

textcolor

Funktion Legt die Zeichenfarbe für folgende Textausgaben fest.

Syntax

```
#include <conio.h>
void textcolor(int newcolor);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *textcolor* legt die Zeichenfarbe für folgende Ausgaben fest, der zulässige Wertebereich für *newcolor* ist 0 bis 15 oder einer der in conio.h definierten symbolischen Konstanten. Falls Sie symbolische Konstanten verwenden, muß die Include-Datei conio.h eingebunden werden.

Wenn *textcolor* aufgerufen wurde, verwenden alle folgenden Aufrufe von Funktionen mit direkter Bildschirmausgabe (wie z.B. *cprintf*) den Parameter *newcolor*. Bereits auf dem Bildschirm befindliche Zeichen bleiben unverändert.

Die folgenden symbolischen Konstanten für die Zeichenfarbe sind in conio.h definiert:

Name	Wert	Beschreibung
BLACK	0	schwarz
BLUE	1	blau
GREEN	2	grün
CYAN	3	türkis
RED	4	rot
MAGENTA	5	violett
BROWN	6	braun
LIGHTGRAY	7	hellgrau
DARKGRAY	8	dunkelgrau
LIGHTBLUE	9	hellblau
LIGHTGREEN	10	hellgrün
LIGHTCYAN	11	helltürkis
LIGHTRED	12	hellrot
LIGHTMAGENTA	13	hellviolett
YELLOW	14	gelb
WHITE	15	weiß
BLINK	128	»blinkend«

Durch Addition des Wertes 128 zur Vordergrundfarbe können Sie Zeichen blinkend darstellen lassen. Zu diesem Zweck wurde auch die Konstante BLINK definiert. Sie verwenden sie wie folgt:

```
textcolor (CYAN + BLINK);
```

Hinweis: Nicht alle Farbmonitore reagieren auf das »Intensitätsbit«, das die Farben 0 bis 7 und 8 bis 15 voneinander unterscheidet.

Wie »Farben« auf Systemen mit monochromen, LCD- und Plasma-Bildschirmen dargestellt werden (Helligkeitsgrade, spezielle Muster oder Attribute wie unterstrichen, fett, etc.), hängt von der jeweiligen Hardware ab.

Rückgabewert Keiner.

Siehe auch *gettextinfo, highvideo, lowvideo, normvideo, textattr, textbackground*

Beispiel

```
#include <conio.h>

int main(void)
{
   int i;

   for (i=0; i<15; i++)
   {
       textcolor(i);
       cprintf("Foreground Color\r\n");
```

```
    }
    return 0;
}
```

textheight

Funktion

Liefert die Höhe eines Strings in Pixel zurück.

Syntax

```
#include <graphics.h>
int far textheight(char far *textstring);
```

DOS	UNIX	Windows	ANSI C	nur C++
	x			

Beschreibung

textheight berechnet aus dem als *textstring* übergebenen String sowie dem aktuellen Font und dem Vergrößerungsfaktor den vertikalen Platzbedarf einer Textzeile.

Diese Funktion ist sehr hilfreich zur Berechnung von Zeichenabständen, der Höhe eines Zeichenfensters, der Größe von Texten, die in eine Grafik oder ein kleines Fenster eingepaßt werden sollen, usw.

Wenn beispielsweise der Standardfont (8 * 8 Pixel) zusammen mit einem Vergrößerungsfaktor von 1 (durch *settextstyle*) gesetzt ist, dann liefert *textheight* für den String *TurboC++* das Ergebnis 8 zurück.

Hinweis: Der Platzbedarf von Strings sollte *grundsätzlich* über *textheight* und *textwidth* ermittelt werden. Programme, die diese Funktionen anstelle von Konstanten verwenden, erfordern bei einem Wechsel der Schrift oder des Grafiktreibers keine Modifikationen.

Rückgabewert

Die Funktion liefert die Texthöhe in Pixel zurück.

Siehe auch

gettextsettings, outtext, outtextxy, settextstyle, textwidth

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int y = 0;
   int i;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
```

```
   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   /* draw some text on the screen */
   for (i=1; i<11; i++)
   {
      /* select the text style, direction, and size */
      settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

      /* create a message string */
      sprintf(msg, "Size: %d", i);

      /* output the message */
      outtextxy(1, y, msg);

      /* advance to the next text line */
      y += textheight(msg);
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

textmode

Funktion

Setzt einen Textmodus.

Syntax

```
#include <conio.h>
void textmode(int newmode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung

textmode setzt den durch *newmode* angegebenen Textmodus, löscht den Bildschirm, erweitert das Textfenster auf die volle Bildschirmgröße und setzt das Attribut für folgende Zeichenausgaben über einen Aufruf von *normvideo*.

Der Textmodus (Parameter *newmode*) wird über eine symbolische Konstante des Aufzählungstyps *text_modes* (definiert in conio.h) angegeben. Falls Sie diese symbolischen Konstanten

verwenden, muß die Include-Datei conio.h eingebunden werden.

Die folgenden Konstanten sind in *text_modes* definiert:

Name	Wert	Beschreibung
LASTMODE	-1	»vorheriger« Textmodus
BW40	0	schwarz-weiß, 40 Spalten
C40	1	farbig, 40 Spalten
BW80	2	schwarz-weiß, 80 Spalten
C80	3	farbig, 80 Spalten
MONO	7	schwarz-weiß, 80 Spalten für Monochrom-Adapter bzw. Hercules-Karte
C4350	64	EGA 43-Zeilen und VGA 50-Zeilen-Modus

Ein Aufruf von *textmode* mit der Konstanten LASTMODE als Parameter ist nur dann sinnvoll, wenn der Bildschirm bereits im Textmodus ist, der Bildschirm wird dann auf den zuletzt verwendeten Textmodus umgeschaltet.

textmode sollte generell nur verwendet werden, wenn der Bildschirm bereits im Textmodus ist (um so den Textmodus temporär zu verändern). Wenn von »Grafik« auf »Text« zurückgeschaltet werden soll, empfiehlt sich die Verwendung von *restorecrtmode*.

Rückgabewert Keiner.

Siehe auch *gettextinfo, window*

Beispiel

```
#include <conio.h>

int main(void)
{
   textmode(BW40);
   cprintf("ABC");
   getch();

   textmode(C40);
   cprintf("ABC");
   getch();

   textmode(BW80);
   cprintf("ABC");
   getch();

   textmode(C80);
   cprintf("ABC");
   getch();
```

```
   textmode(MONO);
   cprintf("ABC");
   getch();

   return 0;
}
```

textwidth

Funktion — Liefert die Breite eines Strings in Pixel zurück.

Syntax

```
#include <graphics.h>
int far textwidth(char far *textstring);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung — *textwidth* berechnet aus dem als *textstring* übergebenen String sowie dem aktuellen Font und dem Vergrößerungsfaktor den horizontalen Platzbedarf einer Textzeile. Wenn beispielsweise der Standardfont (8 * 8 Pixel) zusammen mit dem Vergrößerungsfaktor 1 gesetzt ist, dann liefert *textwidth* für den String *Turbo* das Ergebnis 40 (=5 * 8) zurück.

Hinweis: Der Platzbedarf von Strings sollte *grundsätzlich* über *textheight* und *textwidth* ermittelt werden. Nicht nur, daß das Zentrieren von Titeln, Einpassen von Beschriftungen usw. auf diese Weise wesentlich einfacher geht – Programme, die diese Funktionen anstelle von Konstanten verwenden, erfordern bei einem Wechsel der Schrift oder des Grafiktreibers keine Modifikationen.

Rückgabewert — Die Funktion liefert die Textbreite in Pixel zurück.

Siehe auch — *gettextsettings, outtext, outtextxy, settextstyle, textheight*

Beispiel

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int x = 0, y = 0;
   int i;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");
```

```
   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n",
             grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
   }

   y = getmaxy() / 2;

   settextjustify(LEFT_TEXT, CENTER_TEXT);
   for (i=1; i<11; i++)
   {
      /* select the text style, direction, and size */
      settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

      /* create a message string */
      sprintf(msg, "Size: %d", i);

      /* output the message */
      outtextxy(x, y, msg);

      /* advance to the end of the text */
      x += textwidth(msg);
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

time

Funktion: Liefert Datum und Uhrzeit des Systems.

Syntax:

```
#include <time.h>
time_t time(time_t *timer);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung: *time* setzt die mit *timer* übergebene Variable auf die Anzahl der Sekunden, die seit dem 1. Januar 1970, 00:00:00 (Greenwich Mean Time) verstrichen sind. *timer* darf kein Nullzeiger sein.

Rückgabewert *time* liefert die Anzahl der seit dem 1.1.1970 verstrichenen Sekunden zurück.

Siehe auch *asctime*, *ctime*, *difftime*, *ftime*, *gettime*, *gmtime*, *localtime*, *settime*, *stime*, *tzset*

Beispiel

```
#include <time.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   time_t t;

   t = time(NULL);
   printf("The number of seconds since "
          "January 1, 1970 is %ld",t);
   return 0;
}
```

tmpfile

Funktion Öffnet eine temporäre Datei im binären Modus.

Syntax

```
#include <stdio.h>
FILE *tmpfile(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *tmpfile* erzeugt eine temporäre binäre Datei und öffnet diese Datei im Modus »update« (w+b). Diese Datei wird beim Schließen bzw. beim Ende des Programms automatisch wieder gelöscht.

tmpfile legt die temporäre Datei in dem Verzeichnis an, das im *Environment*-Eintrag TMP angegebenen ist. Ist TMP nicht definiert, wird der Eintrag TEMP verwendet. Ist weder TMP noch TEMP definiert, erzeugt *tmpfile* die Datei im aktuellen Verzeichnis.

Rückgabewert Bei fehlerfreier Ausführung liefert *tmpfile* einen Zeiger auf die Dateivariable zurück. Wenn die Datei aus irgendeinem Grund nicht erzeugt oder geöffnet werden konnte, ist das Funktionsergebnis NULL.

Siehe auch *fopen*, *tmpnam*

Beispiel

```
#include <stdio.h>
#include <process.h>
```

```
int main(void)
{
   FILE *tempfp;

   tempfp = tmpfile();
   if (tempfp)
      printf("Temporary file created\n");
   else
   {
      printf("Unable to create temporary file\n");
      exit(1);
   }

   return 0;
}
```

tmpnam

Funktion: Erzeugt eindeutige Namen für temporäre Dateien.

Syntax:

```
#include <stdio.h>
char *tmpnam(char *s);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung: *tmpnam* erzeugt bei jedem Aufruf einen neuen String und kann für die Benennung temporärer Dateien verwendet werden. Die Funktion erzeugt bis zu 65535 verschiedene Dateinamen. Für diese Obergrenze ist in stdio.h eine Konstante mit dem Namen TMP_MAX definiert.

Wenn der als *s* übergebene Zeiger den Wert NULL hat, liefert *tmpnam* einen Zeiger auf einen statischen Puffer zurück, der durch jeden Aufruf dieser Funktion erneut gesetzt wird. Wenn *s* einen Wert ungleich NULL hat, erwartet *tmpnam*, daß dieser Zeiger auf ein **char**-Array mit mindestens *L_tmpnam* Elementen zeigt, und kopiert das Funktionsergebnis in dieses Array. Die Konstante *L_tmpnam* ist in stdio.h definiert.

tmpnam legt die temporäre Datei in dem Verzeichnis an, das im *Environment*-Eintrag TMP angegebenen ist. Ist TMP nicht definiert, wird der Eintrag TEMP verwendet. Ist weder TMP noch TEMP definiert, erzeugt *tmpnam* die Datei im aktuellen Verzeichnis.

Hinweis: tmpnam ist eine reine Stringfunktion und hat deshalb keinen Einfluß auf die Dateibearbeitung selbst (vgl. *tmpfile*). Es liegt also in der Verantwortung des Programmierers, temporär erzeugte Dateien vor dem Ende eines Programms wieder zu löschen (z.B. mit *remove*).

Rückgabewert Wenn *s* den Wert NULL hat, liefert *tmpnam* einen Zeiger auf einen internen statischen Puffer, ansonsten wird *s* zurückgeliefert.

Siehe auch *tmpfile*

Beispiel

```
#include <stdio.h>

int main(void)
{
   char name[13];

   tmpnam(name);
   printf("Temporary name: %s\n", name);
   return 0;
}
```

toascii

Funktion Konvertiert Zeichen in das ASCII-Format.

Syntax

```
#include <ctype.h>
int toascii(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *toascii* ist ein Makro, das den übergebenen Wert *c* (durch Löschen aller Bits, außer der 7 niederwertigen) auf den Bereich 0 bis 127 begrenzt und das Ergebnis dieser Operation zurückliefert.

Rückgabewert Das Makro liefert das Ergebnis der Konvertierung von *c* als Wert vom Typ **int** zurück.

Beispiel

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  int number, result;
  number = 511;
  result = toascii(number);
  printf("%d %d\n", number, result);
  return 0;
}
```

_tolower

Funktion Wandelt Groß- in Kleinbuchstaben um.

Syntax
```
#include <ctype.h>
int _tolower(int ch);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *_tolower* ist ein Makro, das wie *tolower* das übergebene Zeichen *ch* in einen Kleinbuchstaben verwandelt. Dieses Makro ist jedoch *nur* auf Großbuchstaben (A bis Z) anwendbar – für Ziffern und Sonderzeichen erzeugt es unsinnige Ergebnisse. *_tolower* kann nur verwendet werden, wenn die Include-Datei ctype.h eingebunden ist.

Rückgabewert Das Makro liefert als Ergebnis der Konvertierung einen Wert vom Typ **int** zurück, falls *ch* ein Großbuchstabe ist; ansonsten ist das Resultat undefiniert.

Beispiel
```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   int length, i;
   char *string = "THIS IS A STRING.";

   /* We should be checking each character to
   make sure it is an uppercase before passing
   it to _tolower! The result of passing it a
   non-uppercase is undefined. */

   length = strlen(string);
   for (i = 0; i < length; i++) {
      string[i] = _tolower(string[i]);
   }
   printf("%s\n",string);
   return 0;
}
```

tolower

Funktion Wandelt Groß- in Kleinbuchstaben um.

Syntax
```
#include <ctype.h>
int tolower(int ch);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *tolower* ist eine Funktion, die das als Integer übergebene Zeichen *ch* in einen Kleinbuchstaben umwandelt. Damit diese Funktion korrekt arbeitet, muß *ch* einen Wert im Bereich von EOF bis 255 haben.

Rückgabewert Die Funktion liefert als Ergebnis der Konvertierung einen Wert vom Typ **int** zurück, falls der zu konvertierende Wert von *ch* ein Großbuchstabe ist; ansonsten wird das Zeichen unverändert zurückgeliefert.

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   int length, i;
   char *string = "THIS IS A STRING";

   length = strlen(string);
   for (i=0; i<length; i++)
   {
       string[i] = tolower(string[i]);
   }
   printf("%s\n",string);

   return 0;
}
```

_toupper

Funktion Wandelt Klein- in Großbuchstaben um.

Syntax

```
#include <ctype.h>
int _toupper(int ch);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *_toupper* ist ein Makro und das Gegenstück zu *_tolower*. Es wandelt wie die Funktion *toupper* Kleinbuchstaben in Großbuchstaben um, kann aber nur zusammen mit Kleinbuchstaben (a bis z) benutzt werden. Die Anwendung auf Sonderzeichen, Ziffern, Umlaute etc. erzeugt unsinnige Ergebnisse). *_toupper* kann nur verwendet werden, wenn die Include-Datei ctype.h eingebunden ist.

Rückgabewert Das Makro liefert als Ergebnis der Konvertierung einen Wert vom Typ **int** zurück, falls der zu konvertierende Wert von *ch* ein Kleinbuchstabe ist; ansonsten ist das Resultat undefiniert.

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   int length, i;
   char *string = "this is a string";

   /*
      We should be checking each character to make
      sure it is lowercase before passing it to
      _toupper. The result passing a non-lowercase
      is undefined.
   */

   length = strlen(string);
   for (i = 0; i < length; i++) {
      string[i] = _toupper(string[i]);
   }
   printf("%s\n",string);
   return 0;
}
```

toupper

Funktion Wandelt Klein- in Großbuchstaben um.

Syntax

```
#include <ctype.h>
int toupper(int ch);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *toupper* ist eine Funktion, die das als Integer übergebene Zeichen *ch* in einen Großbuchstaben verwandelt. Damit diese Funktion korrekt arbeitet, muß *ch* einen Wert im Bereich von EOF bis 255 haben. Andere Zeichen werden von *toupper* nicht berücksichtigt.

Rückgabewert Die Funktion liefert als Ergebnis der Konvertierung ein **int** zurück, falls der zu konvertierende Wert von *ch* ein Kleinbuchstabe ist; ansonsten wird das Zeichen unverändert zurückgeliefert.

Beispiel

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   int length, i;
   char *string = "this is a string";

   length = strlen(string);
   for (i=0; i<length; i++)
   {
      string[i] = toupper(string[i]);
   }

   printf("%s\n",string);

   return 0;
}
```

tzset

Funktion Setzt die globalen Variablen *daylight*, *timezone* und *tzname*.

Syntax

```
#include <time.h>
void tzset(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *tzset* ist auf XENIX-Systemen verfügbar.

Die Funktion setzt die globalen Variablen *daylight*, *timezone* und *tzname* basierend auf dem *Environment*-Eintrag TZ. Die Bibliotheksfunktionen *ftime* und *localtime* verwenden diese globalen Variablen, um die Greenwich Mean Time (GMT) auf die lokale Zeitzone zu setzen. *tzset* erwartet einen Eintrag in der Tabelle *Environment* des Betriebssystems, der den Namen *TZ* und das folgende Format haben muß:

```
TZ = zzz[+|-]d[d][lll]
```

Das Feld *zzz* steht für den Namen der Zeitzone, der aus drei Buchstaben bestehen muß (z.B. »MEZ« für die mitteleuropäische Zeit).

Das ebenfalls obligatorische Feld *[+|-]d[d]* steht für die Abweichung von der Weltstandardzeit (»Greenwich Mean Time« – GMT) in Stunden, die durch einen Zahlenwert mit mindestens einer Ziffer (und falls notwendig, einem Vorzeichen) ausgedrückt wird. Die Verschiebung für Länder westlich des 0.

Längengrades wird mit positiven Werten notiert (+5 entspricht beispielsweise der Zeitverschiebung von New York gegenüber Greenwich), für Zeitzonen östlich von Greenwich entstehen negative Werte (für die MEZ gilt beispielsweise -1).

Dieser Wert wird zur Berechnung der globalen Variablen *timezone* verwendet. *timezone* repräsentiert die Zeitdifferenz in Sekunden zwischen der Weltstandardzeit (GMT) und der Ortszeit.

Das Feld *lll* ist im Gegensatz zu den restlichen Feldern optional: wenn es angegeben ist, muß es aus drei Buchstaben bestehen und legt den Namen der lokalen »Sommerzeitzone« fest (z.B. »MES« für mitteleuropäische Sommerzeit). *tzset* setzt die globale Variable *daylight* abhängig vom Vorhandensein dieses Feldes entweder auf 0 (= keine Sommerzeit) oder auf einen Wert ungleich Null (Sommerzeitzone definiert).

Wenn *tzset* keinen *Environment*-Eintrag des Namens TZ findet oder dieser Eintrag nicht den zuvor gezeigten Regeln genügt, verwendet die Funktion den String »EST5EDT« (entspricht »Eastern Standard Time«, Zeitverschiebung +5 Stunden gegenüber GMT, Sommerzeit definiert) und setzt die globalen Variablen *daylight*, *timezone* und *tzname* mit den entsprechenden Werten.

Die globale Variable *tzname*[*0*] zeigt auf einen drei Zeichen langen String, der den Namen der Zeitzone aus dem *Environment*-String TZ speichert. *tzname*[*1*] zeigt auf einen drei Zeichen langen String, der den Namen der Sommerzeitzone aus dem *Environment*-String TZ speichert. Wurde keine Sommerzeitzone genannt, zeigt *tzname*[*1*] auf einen Null-String.

Rückgabewert Keiner.

Siehe auch *asctime*, *ctime*, *ftime*, *gmtime*, *localtime*, *stime*, *time*

Beispiel

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   time_t td;

   putenv("TZ=PST8PDT");
   tzset();
   time(&td);
   printf("Current time = %s\n",
          asctime(localtime(&td)));
   return 0;
}
```

ultoa

Funktion — Konvertiert einen Wert vom Typ **unsigned long** in einen String.

Syntax

```
#include <stdlib.h>
char *ultoa(unsigned long value, char *string,
            int radix);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung — Die Funktion konvertiert den übergebenen Wert *value* in einen nullterminierten String und speichert das Ergebnis in der durch *string* bezeichneten Variablen. *ultoa* erwartet einen Wert vom Typ **unsigned long** als *value*. Der Parameter *radix* legt die Basis fest, zu der die Konvertierung stattfinden soll – mögliche Werte liegen im Bereich von 2 bis 36.

ultoa prüft nicht auf Überlauf; wenn *radix* den Wert 10 hat und *value* kleiner als Null ist, stellt *ultoa* im Unterschied zu *itoa* und *ltoa* dem Ergebnis kein Minuszeichen voran.

Hinweis: Der übergebene *string* muß in allen Fällen lang genug sein, um das Ergebnis der Konvertierung (inklusive des abschließenden Nullzeichens \0) aufnehmen zu können. *ultoa* kann maximal 33 Zeichen zurückgeben.

Rückgabewert — Die Funktion liefert *string* zurück.

Siehe auch — *itoa, ltoa*

Beispiel

```
#include <stdlib.h>
#include <stdio.h>

int main( void )
{
   unsigned long lnumber = 3123456789L;
   char string[25];

   ultoa(lnumber,string,10);
   printf("string = %   s  unsigned long = %   lu\n",
          string,lnumber);
   return 0;
}
```

umask

Funktion — Setzt das Bitfeld für Dateizugriffe (Lese- und Schreiboperationen).

Syntax

```
#include <io.h>
unsigned umask(unsigned mode);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

umask setzt das Bitfeld, das für Aufrufe von *open* und *creat* gilt. Die in *mode* gesetzten Bits werden in den Zugriffsberechtigungen aller im folgenden mit *create* und *open* erzeugten Dateien gelöscht.

Der Parameter *mode* kann über die folgenden symbolischen Konstanten festgelegt werden, die in sys/stat.h definiert sind:

S_IWRITE	Schreiben erlaubt
S_IREAD	Lesen erlaubt
S_IREAD | S_IWRITE	Lesen und Schreiben erlaubt

Rückgabewert

Bei fehlerfreier Ausführung wird der vorherige Wert des Bitfelds zurückgeliefert. Im Fehlerfall hat *umask* keinen Rückgabewert.

Siehe auch

creat, open

Beispiel

```
#include <io.h>
#include <stdio.h>
#include <sys\stat.h>

#define FILENAME "TEST.$$$"

int main(void)
{
    unsigned oldmask;

    FILE *f;
    struct stat statbuf;

    /* Cause subsequent files to be created
       as read-only */
    oldmask = umask(S_IWRITE);
    printf("Old mask = 0x%x\n",oldmask);

    /* Create a zero-length file */
    if ((f = fopen(FILENAME,"w")) == NULL)
    {
        perror("Unable to create output file");
        return (1);
    }
    fclose(f);

    /* Verify that the file is read-only */
    if (stat(FILENAME,&statbuf) != 0)
    {
        perror("Unable to get information about "
               "output file");
        return (1);
```

```
    }
    if (statbuf.st_mode & S_IWRITE)
        printf("Error! %s is writable!\n",FILENAME);
    else
        printf("Success! %s is not writable.\n",
               FILENAME);
    return (0);
}
```

ungetc

Funktion — Stellt ein aus einem *stream* gelesenes Zeichen zurück.

Syntax

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *ungetc* stellt ein Zeichen für eine erneute Leseoperation zurück in den Eingabe-Stream *stream,* der für Leseoperationen geöffnet sein muß. Ein Zeichen kann jederzeit zurückgestellt und später mit *getc* oder *fread* wieder gelesen werden. Es wird immer nur ein Zeichen gespeichert – ein zweiter Aufruf von *ungetc* ohne ein dazwischenliegendes *getc* überschreibt das zuvor zurückgestellte Zeichen.

Die Anwendung von *fflush, fseek, fsetpos* oder *rewind* auf eine Datei löscht deren Puffer und damit auch ein eventuell mit *ungetc* zurückgestelltes Zeichen.

Rückgabewert — *ungetc* liefert das zurückgestellte Zeichen als Funktionsergebnis, wenn die Aktion fehlerfrei ausgeführt werden konnte, ansonsten EOF.

Siehe auch — *fgetc, getc, getchar*

Beispiel

```
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   int i=0;
   char ch;

   puts("Input an integer followed by a char:");

   /* read chars until non digit or EOF */
   while((ch = getchar()) != EOF && isdigit(ch))
      i = 10 * i + ch - 48;
          /* convert ASCII into int value */
```

```
   /* if non digit char was read, push it
      back into input buffer */
   if (ch != EOF)
      ungetc(ch, stdin);

   printf("i = %  d, next char in buffer = %    c\n",
          i, getchar());
   return 0;
}
```

ungetch

Funktion

Stellt ein von der Tastatur gelesenes Zeichen in den Tastaturpuffer zurück.

Syntax

```
#include <conio.h>
int ungetch(int c);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

ungetch stellt das Zeichen *ch* für die nächste Leseaktion von der Tastatur zurück. Es kann nur jeweils ein Zeichen gespeichert werden – *ungetch* überschreibt ein zuvor zurückgestelltes Zeichen aber nicht, sondern liefert einen Fehler.

Rückgabewert

Die Funktion liefert das Zeichen *ch* zurück; bei einem Fehler wird EOF zurückgeliefert.

Siehe auch

getch, getche

Beispiel

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>

int main( void )
{
   int i=0;
   char ch;

   puts("Input an integer followed by a char:");

   /* read chars until non digit or EOF */
   while((ch = getche()) != EOF && isdigit(ch))
      i = 10 * i + ch - 48;
          /* convert ASCII into int value */

   /* if non digit char was read, push it
      back into input buffer */
   if (ch != EOF)
      ungetch(ch);
```

```
   printf("\n\ni = %d, next char in buffer = %c\n",
          i, getch());
   return 0;
}
```

unixtodos

Funktion

Konvertiert Datum und Uhrzeit aus dem UNIX- in das DOS-Format.

Syntax

```
#include <dos.h>
void unixtodos(long time, struct date *d,
               struct time *t);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

unixtodos erwartet einen Wert vom Typ **long**, der Datum und Uhrzeit im UNIX-Format enthält, und setzt zwei Strukturen des Typs *date* bzw. *time*, auf die *d* und *t* zeigen.

time darf keine Uhrzeit von dem 1. Januar 1980 00:00:00 sein.

Rückgabewert

Keiner.

Siehe auch

dostounix

Beispiel

```
#include <stdio.h>
#include <dos.h>

char *month[] = {"---", "Jan", "Feb", "Mar", "Apr",
                 "May", "Jun", "Jul", "Aug", "Sep",
                 "Oct", "Nov", "Dec"};

#define SECONDS_PER_DAY 86400L
          /* the number of seconds in one day */

struct date dt;
struct time tm;

int main(void)
{
   unsigned long val;

/* get today's date and time */
   getdate(&dt);
   gettime(&tm);
   printf("today is %d %s %d\n", dt.da_day,
          month[dt.da_mon], dt.da_year);

/* convert date and time to unix format
   (num of seconds since Jan 1, 1970 */
```

```
   val = dostounix(&dt, &tm);
/* subtract 42 days worth of seconds */
   val -= (SECONDS_PER_DAY * 42);

/* convert back to dos time and date */
   unixtodos(val, &dt, &tm);
   printf("42 days ago it was %d %s %d\n",
          dt.da_day, month[dt.da_mon], dt.da_year);
   return 0;
}
```

unlink

Funktion — Löscht eine Datei.

Syntax

```
#include <io.h>
int unlink(const char *filename);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung — *unlink* löscht die durch *filename* angegebene Datei, wobei *filename* einen vollständigen Suchweg und/oder einen Laufwerksbezeichner enthalten kann. Die Jokerzeichen * und ? sind in *filename* nicht erlaubt.

Schreibgeschützte Dateien können mit *unlink* nicht gelöscht werden. Um eine schreibgeschützte Datei zu löschen, muß zuerst das entsprechende Attribut mit *chmod* oder *_chmod* entfernt werden; das Löschen von Verzeichnissen ist nur über die Funktion *rmdir* möglich.

Hinweis: Eine geöffnete Datei muß vor dem Löschen geschlossen werden.

Rückgabewert — Bei fehlerfreier Ausführung liefert *unlink* den Wert 0 zurück. Im Falle eines Fehlers ist das Funktionsergebnis -1 und *errno* bekommt einen der folgenden Werte:

ENOENT	Suchweg/Datei nicht gefunden
EACCES	Zugriff nicht gestattet

Siehe auch — *chmod, remove*

Beispiel

```
#include <stdio.h>
#include <io.h>

int main(void)
{
   FILE *fp = fopen("junk.jnk","w");
   int status;
```

```
   fprintf(fp,"junk");

   status = access("junk.jnk",0);
   if (status == 0)
      printf("File exists\n");
   else
      printf("File doesn't exist\n");

   fclose(fp);
   unlink("junk.jnk");
   status = access("junk.jnk",0);
   if (status == 0)
      printf("File exists\n");
   else
      printf("File doesn't exist\n");

   return 0;
}
```

unlock

Funktion Entfernt mit *lock* gesetzte Sperren bei Dateien, die von mehreren Programmen gleichzeitig benutzt werden (SHARED-Modus).

Syntax

```
#include <io.h>
int unlock(int handle, long offset, long length);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung Ab der Version 3.0 von DOS ist der Modus SHARED definiert, in dem eine Datei simultan von mehreren Programmen genutzt werden kann.

unlock gibt einen vorher mit *lock* gesperrten Bereich wieder frei. Hier ist zu beachten: Vor dem Schließen der Datei müssen sämtliche Bereiche wieder entsperrt werden und ein Programm muß alle Sperren aufheben, bevor die Ausführung beendet wird.

Rückgabewert Die Funktion liefert bei fehlerfreier Ausführung den Wert 0, ansonsten den Wert -1 zurück.

Siehe auch *lock, sopen*

Beispiel

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
```

```
#include <share.h>
#include <stdio.h>

int main(void)
{
   int handle, status;
   long length;

   handle = sopen("c:\\autoexec.bat",
                  O_RDONLY, SH_DENYNO, S_IREAD);

   if (handle < 0)
   {
       printf("sopen failed\n");
       exit(1);
   }

   length = filelength(handle);
   status = lock(handle,0L,length/2);

   if (status == 0)
      printf("lock succeeded\n");
   else
      printf("lock failed\n");

   status = unlock(handle,0L,length/2);

   if (status == 0)
      printf("unlock succeeded\n");
   else
      printf("unlock failed\n");

   close(handle);
   return 0;
}
```

utime

Funktion Setzt Datum und Uhrzeit einer Datei.

Syntax
```
#include <utime.h>
int utime(char *path, struct utimbuf *times);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *utime* setzt Uhrzeit und Datum für die Datei *path*. Die Modifikationszeit ist in der durch *times* übergebenen Struktur vom

Typ *utimbuf* enthalten. Diese Struktur ist in utime.h definiert und hat das folgende Format:

```
struct utimebuf {
   time_t actime;        /* Zugriffszeit */
   time_t modtime;       /* Modifikationszeit */
};
```

Das Dateisystem von DOS unterstützt nur die Modifikationszeit. Unter DOS ignoriert deshalb ein *utime*-Aufruf *actime* und verwendet nur *modtime* zum Setzen der Modifikationszeit.

Hat *times* den Wert NULL, wird die Modifikationszeit der Datei auf die aktuelle Uhrzeit gesetzt.

Rückgabewert

utime liefert bei fehlerfreier Ausführung den 0 zurück. Im Fehlerfall wird der Wert -1 zurückgeliefert und *errno* bekommt einen der folgenden Werte:

EACCES	Zugriff nicht gestattet
EMFILE	Zu viele Dateien bereits offen
ENOENT	Datei oder Verzeichnis nicht gefunden

Siehe auch

setftime, stat, time

Beispiel

```
/* Copy timestamp from one file to another */
#include <sys\stat.h>
#include <utime.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
  struct stat src_stat;
  struct utimbuf times;
  if(argc != 3) {
    printf( "Usage: copytime <source file> "
           "<dest file>\n" );
    return 1;
  }

  if (stat(argv[1],&src_stat) != 0) {
    perror("Unable to get status of source file");
    return 1;
  }

  times.modtime = times.actime = src_stat.st_mtime;
  if (utime(argv[2],&times) != 0) {
    perror("Unable to set time of destination file");
    return 1;
  }
  return 0;
}
```

va_arg, va_end, va_start

Funktion Makros für Funktionen mit variabler Parameterzahl.

Syntax
```
#include <stdarg.h>
void va_start(va_list ap, lastfix);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung Verschiedene C-Funktionen (wie *vfprintf* und *vprintf*) erwarten einen oder mehrere obligatorische Parameter (wie ein »Ziel« und einen Format-String) und darüber hinaus eine variable Anzahl weiterer Argumente. Die Makros *va_arg*, *va_end* und *va_start* ermöglichen es, Deklarationen und Aufrufe von Funktionen mit variabler Parameterzahl in portierbarer Form zu halten.

Es wird in jedem Fall davon ausgegangen, daß die entsprechende Funktion eine bestimmte Anzahl obligatorischer Parameter erwartet. Zusätzliche Parameter können schrittweise erfaßt werden – auch dann, wenn die Funktion selbst nicht »weiß«, wieviele weitere Angaben zu bearbeiten sind.

In der Datei stdarg.h sind ein Typ (*va_list*) und drei Makros (*va_start*, *va_arg* und *va_end*) deklariert: *va_list* ist der Array-Typ, der die von *va_arg* und *va_end* benötigten Informationen speichert. Eine Funktion mit einer variablen Argumentenliste deklariert eine Variable *ap* vom Typ *va_list*.

va_start: Dieses Makro erwartet die Angabe eines Zeigers des Typs *va_list* (*ap*) sowie den Namen des letzten fixen Parameters der aufzurufenden Funktion (*lastfix*) und setzt *ap* auf den Beginn des ersten variablen Parameters der Funktion.

va_arg: Dieses Makro erwartet einen (zuvor mit *va_start* initialisierten) Zeiger des Typs *va_list* (*ap*) sowie einen Variablentyp (*type*), interpretiert den jeweils nächsten variablen Parameter der Funktion als Wert des angegebenen Typs und liefert ihn zurück. Danach wird der Zeiger um **sizeof**(*type*) erhöht.

Hinweis: Mit *va_arg* können Sie keine Werte vom Typ **char**, **unsigned char** oder **float** als Argumente verwenden.

Der erste Aufruf von *va_art* liefert den Wert des ersten variablen Arguments zurück und erhöht *ap* entsprechend, so daß *ap* auf das nachfolgende Argument zeigt. Weitere Aufrufe von *va_arg* – wieder mit *ap* – liefern die jeweils nachfolgenden Argumente zurück.

va_end: Dieses Makro hilft der aufgerufenen Funktion, einen normalen Rücksprung durchzuführen. *va_end* könnte *ap* ver-

ändern und damit bis zu einem nächsten Aufruf von *va_start* unbrauchbar machen. *va_end* sollte aufgerufen werden, nachdem *va_arg* alle Argumente abgearbeitet hat; wird dies unterlassen, resultiert unter Umständen ein undefiniertes Verhalten des Programms.

Rückgabewert *va_start* und *va_end* haben keinen Rückgabewert, *va_arg* liefert das aktuelle Argument der Funktion in dem durch *type* angegebenen Format zurück.

Siehe auch *v...printf, v...scanf*

Beispiel

```
#include <stdio.h>
#include <stdarg.h>

/* calculate sum of a 0 terminated list */
void sum(char *msg, ...)
{
   int total = 0;
   va_list ap;
   int arg;
   va_start(ap, msg);
   while ((arg = va_arg(ap,int)) != 0) {
      total += arg;
   }
   printf(msg, total);
   va_end(ap);
}

int main(void) {
   sum("The total of 1+2+3+4 is %d\n", 1,2,3,4,0);
   return 0;
}
```

Programmausgabe:

```
The total of 1+2+3+4 is 10
```

vfprintf

Funktion Formatierte Ausgabe in einen Stream.

Syntax

```
#include <stdio.h>
int vfprintf(FILE *stream, const char *format,
             va_list arglist);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung *vfprintf* ist unter UNIX System V verfügbar.

Die *v...printf*-Funktionen akzeptieren im Unterschied zu

ihren *...printf*-Gegenstücken die Übergabe der Parameter über einen Zeiger auf eine Argumentenliste anstelle einer Argumentenliste.

vfprintf akzeptiert einen Zeiger auf ein Reihe von Argumenten, wendet auf jedes Argument einen Formatbezeichner an, der in dem Formatstring enthalten ist, auf den *format* zeigt, und schreibt die formatierten Daten in einen Stream. Die Anzahl der Argumente und Formatbezeichner muß identisch sein.

Hinweis: Weitere Erläuterungen zur Formatierung finden Sie unter der Beschreibung von *printf*.

Rückgabewert Die Funktion liefert die Anzahl der ausgegebenen Bytes zurück. Bei einem Fehler liefert *vfprint* EOF zurück.

Siehe auch *printf, va_arg, va_end, va_start*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

FILE *fp;

int vfpf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vfprintf(fp, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
   char string[4] = "abc";

   fp = tmpfile();
   if (fp == NULL)
   {
      perror("tmpfile() call");
      exit(1);
   }

   vfpf("%d %f %s", inumber, fnumber, string);
   rewind(fp);
   fscanf(fp,"%d %  f % s", &inumber, &fnumber, string);
```

```
    printf("%d %f %s\n", inumber, fnumber, string);
    fclose(fp);

    return 0;
}
```

vfscanf

Funktion Formatierte Eingabe aus einem Stream.

Syntax

```
#include <stdio.h>
int vfscanf(FILE *stream, const char *format,
            va_list arglist);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *vfscanf* ist unter UNIX System V verfügbar.

Die *v...scanf*-Funktionen akzeptieren im Unterschied zu ihren *...scanf*-Gegenstücken die Übergabe der Parameter über einen Zeiger auf eine Argumentenliste anstelle einer Argumentenliste.

vfscanf liest eine Reihe von Eingabefeldern zeichenweise aus einem Stream. Auf jedes Feld wird dann die entsprechende Formatangabe angewendet, die *vfscanf* über den Zeiger *format* auf den Formatstring übergeben wurde. Der Formatstring legt die Anzahl der Eingaben und ihr Format fest. Gelesene Eingaben werden an der Adresse, die in dem *format* nachfolgenden Parameter angegeben wurde, gespeichert: Nach der Analyse der ersten Formatanweisung wird ein Eingabefeld gelesen, konvertiert und an der entsprechenden Adresse gespeichert. Die Anzahl von Eingabefeldern muß der Anzahl von Formatangaben und Adressen genau entsprechen.

Es ist möglich, daß *vfscanf* das Einlesen eines bestimmten Feldes vorzeitig – d.h. bevor das Feldende (markiert durch ein »Whitespace«) erreicht ist – abbricht bzw. den Lesevorgang komplett beendet. In der Beschreibung zu *scanf* werden die Gründe hierfür erläutert.

Hinweis: Weitere Erläuterungen zu Formatbezeichnern finden Sie in der Beschreibung zu *scanf*.

Rückgabewert *vfscanf* liefert die Anzahl der fehlerfrei gelesenen und gespeicherten Felder zurück, wobei nur gelesene Felder nicht gezählt werden. Wenn überhaupt keine Zuweisungen erfolgt sind, liefert *vfscanf* den Wert 0.

Beim Versuch, über das Ende einer Datei hinauszulesen, liefert *vfscanf* Wert EOF zurück.

Siehe auch *fscanf*, *scanf*, *va_arg*, *va_end*, *va_start*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

FILE *fp;

int vfsf(char *fmt, ...)
{
   va_list  argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vfscanf(fp, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
         char string[4] = "abc";

   fp = tmpfile();
   if (fp == NULL)
   {
      perror("tmpfile() call");
      exit(1);
   }
   fprintf(fp,"%d %f %s\n",inumber,fnumber,string);
   rewind(fp);

   vfsf("%d %f %s",&inumber,&fnumber,string);
   printf("%d %f %s\n",inumber,fnumber,string);
   fclose(fp);
   return 0;
}
```

vprintf

Funktion Formatierte Ausgabe zu *stdout*.

Syntax

```
#include <stdarg.h>
int vprintf(const char *format, va_list arglist);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung *vprintf* ist unter UNIX System V verfügbar.

Die *v...printf*-Funktionen akzeptieren im Unterschied zu

ihren *...printf*-Gegenstücken die Übergabe der Argumente über einen Zeiger auf eine Argumentenliste anstelle einer Argumentenliste.

vprintf akzeptiert einen Zeiger auf ein Reihe von Argumenten, wendet auf jedes Argument einen Formatbezeichner an, der in dem Formatstring enthalten ist, auf den *format* zeigt und schreibt die formatierten Daten in *stdout*. Die Anzahl der Argumente und Formatbezeichner muß identisch sein.

Hinweis: Weitere Erläuterungen zur Formatierung finden Sie unter der Beschreibung von *printf*.

Achtung: Wenn Sie in der Option *Compiler/CodeGeneration* das *SS!=DS*-Flag markiert haben, geht *vprintf* davon aus, daß sich die übergebene Adresse im SS-Segment befindet.

Rückgabewert Die Funktion liefert die Anzahl der ausgegebenen Zeichen zurück. Bei einem Fehler liefert *vprintf* den Wert EOF zurück.

Siehe auch *printf, va_arg, va_end, va_start*

Beispiel

```
#include <stdio.h>
#include <stdarg.h>

int vpf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vprintf(fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
   char *string = "abc";

   vpf("%d %f %s\n",inumber,fnumber,string);

   return 0;
}
```

vscanf

Funktion Formatierte Eingabe von *stdin*.

Syntax

```
#include <stdarg.h>
int vscanf(const char *format, va_list arglist);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x			

Beschreibung

vprintf ist unter UNIX System V verfügbar.

Die *v...scanf*-Funktionen akzeptieren im Unterschied zu ihren *...scanf*-Gegenstücken die Übergabe der Argumente über einen Zeiger auf eine Argumentenliste anstelle einer Argumentenliste.

vscanf liest eine Reihe von Eingabefeldern zeichenweise aus der Datei *stdin*. Auf jedes Feld wird dann der entsprechende Formatbezeichner angewendet, der *vscanf* über den Zeiger *format* auf den Formatstring übergeben wurde. Der Formatstring legt die Anzahl der Eingaben und ihr Format fest. Gelesene Eingaben werden an der Adresse, die in dem *format* nachfolgenden Parameter angegeben wurde, gespeichert: Nach der Analyse der ersten Formatanweisung wird ein Eingabefeld gelesen, konvertiert und an der entsprechenden Adresse gespeichert. Die Anzahl von Eingabefeldern muß der Anzahl von Formatangaben und Adressen genau entsprechen.

Es ist möglich, daß *vscanf* das Einlesen eines bestimmten Feldes vorzeitig – d.h. bevor das Feldende (markiert durch ein »Whitespace«) erreicht ist – abbricht bzw. den Lesevorgang komplett beendet. In der Beschreibung zu *scanf* werden die Gründe hierfür erläutert.

Hinweis: Weitere Erläuterungen zu Formatbezeichnern finden Sie in der Beschreibung zu *scanf*.

Rückgabewert

vscanf liefert die Anzahl der fehlerfrei gelesenen, konvertierten und gespeicherten Felder zurück, wobei Felder, die nur gelesen wurden, nicht gezählt werden. Wenn überhaupt keine Zuweisungen erfolgt sind, ist der Rückgabewert 0.

Beim Versuch, über das Ende der Datei hinauszulesen, liefert *vscanf* den Wert EOF zurück.

Siehe auch

fscanf, *scanf*, *va_arg*, *va_end*, *va_start*

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

int vscnf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   printf("Enter an integer, a float, and a string "
          "(e.g. i,f,s,)\n");
   va_start(argptr, fmt);
   cnt = vscanf(fmt, argptr);
```

```
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber;
   float fnumber;
   char string[80];

   vscnf("%d, %f, %s", &inumber, &fnumber, string);
   printf("%d %f %s\n", inumber, fnumber, string);

   return 0;
}
```

vsprintf

Funktion: Formatierte Ausgabe in einen String.

Syntax:

```
#include <stdarg.h>
int vsprintf(char *buffer, const char *format,
             va_list arglist);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x		x	

Beschreibung: *vsprintf* ist unter UNIX System V verfügbar.

Die v...*printf*-Funktionen akzeptieren im Unterschied zu ihren ...*printf*-Gegenstücken die Übergabe der Argumente über einen Zeiger auf eine Argumentenliste anstelle einer Argumentenliste.

vsprintf akzeptiert einen Zeiger auf ein Reihe von Argumenten, wendet auf jedes Argument einen Formatbezeichner an, der in dem Formatstring enthalten ist, auf den *format* zeigt und schreibt die formatierten Daten in einen String. Die Anzahl der Argumente und Formatbezeichner muß identisch sein.

Hinweis: Weitere Erläuterungen zur Formatierung finden Sie unter der Beschreibung zu *printf*.

Rückgabewert: Die Funktion liefert die Anzahl der ausgegebenen Zeichen zurück. Bei einem Fehler liefert *vsprintf* EOF zurück.

Siehe auch: *printf, va_arg, va_end, va_start*

Beispiel:

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>
```

```
char buffer[80];

int vspf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vsprintf(buffer, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
   char string[4] = "abc";

   vspf("%d %f %s", inumber, fnumber, string);
   printf("%s\n", buffer);
   return 0;
}
```

vsscanf

Funktion Formatierte Eingabe aus einem String.

Syntax
```
#include <stdarg.h>
int vsscanf(const char *buffer, cosnt char *format,
            va_list arglist);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *vsscanf* ist unter UNIX System V verfügbar.

Die v...*scanf*-Funktionen akzeptieren im Unterschied zu ihren ...*scanf*-Gegenstücken die Übergabe der Argumente über einen Zeiger auf eine Argumentenliste anstelle einer Argumentenliste.

vsscanf liest eine Reihe von Eingabefeldern zeichenweise aus einem String. Auf jedes Feld wird dann der entsprechende Formatbezeichner angewendet, der *vsscanf* über den Zeiger *format* auf den Formatstring übergeben wurde. Der Formatstring legt die Anzahl der Eingaben und ihr Format fest. Gelesene Eingaben werden an der Adresse, die in dem *format* nachfolgenden Parameter angegeben wurde, gespeichert: Nach der Analyse der ersten Formatanweisung wird ein Eingabefeld gelesen, konvertiert und an der entsprechenden Adresse

gespeichert. Die Anzahl von Eingabefeldern muß der Anzahl von Formatangaben und Adressen genau entsprechen.

Es ist möglich, daß *vsscanf* das Einlesen eines bestimmten Feldes vorzeitig – d.h. bevor das Feldende (markiert durch ein »Whitespace«) erreicht ist – abbricht bzw. den Lesevorgang komplett beendet.

In der Beschreibung zu *scanf* werden die Gründe hierfür erläutert.

Hinweis: Weitere Erläuterungen zu Formatbezeichnern finden Sie in der Beschreibung zu *scanf*.

Rückgabewert

vsscanf liefert die Anzahl der fehlerfrei gelesenen, konvertierten und gespeicherten Felder zurück, wobei Felder, die nicht gespeichert wurden, nicht gezählt werden. Wenn überhaupt keine Zuweisungen erfolgt sind, ist der Rückgabewert 0.

Beim Versuch, über das Ende des Strings hinauszulesen, wird der Wert EOF zurückgeliefert.

Siehe auch

fscanf, *scanf*, *sscanf*, *va_arg*, *va_end*, *va_start*, *vfscanf*

Beispiel

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

char buffer[80] = "30 90.0 abc";

int vssf(char *fmt, ...)
{
   va_list  argptr;
   int cnt;

   fflush(stdin);
   va_start(argptr, fmt);
   cnt = vsscanf(buffer, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber;
   float fnumber;
   char string[80];

   vssf("%d %f %s", &inumber, &fnumber, string);
   printf("%d %f %s\n", inumber, fnumber, string);
   return 0;
}
```

wcstombs

Funktion — Konvertiert ein *wchar_t*-Array in einen String aus Multibyte-Zeichen.

Syntax

```
#include <stdlib.h>
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — *wcstombs* konvertiert die über *pcws* angegebenen Elemente des Typs *wchar_t* in den String *s*.

Die Konvertierung wird beendet, wenn *wcstombs* auf ein Nullzeichen oder ein ungültiges Multibyte-Zeichen trifft.

Der Parameter *n* legt fest, wieviele Bytes maximal konvertiert werden. Wenn die betreffende Anzahl Bytes konvertiert und kein Nullzeichen angetroffen wurde, ist das Array *s* nicht nullterminiert.

Das Verhalten von *wcstombs* ist von der Einstellung der LC_CTYPE-Kategorie in der aktuellen *locale*-Variablen abhängig.

Rückgabewert — *mbstowcs* liefert *(size_t)-1* zurück, wenn es auf ein ungültiges Multibyte-Zeichen trifft. Anderfalls wird die Anzahl der geänderten Bytes zurückgeliefert (das abschließende Zeichen ist – falls vorhanden – darin nicht enthalten).

wctomb

Funktion — Konvertiert einen Wert des Typs *wchar_t* in ein Multibyte-Zeichen.

Syntax

```
#include <stdlib.h>
int wctomb (char *s, wchar_t wc);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x	x	

Beschreibung — Ist *s* ungleich NULL, stellt *wctomb* die Anzahl der Bytes fest, die zur Darstellung des durch *wc* angegebenen Multibyte-Zeichens erforderlich sind (einschließlich nötiger Änderungen des Status der Umschalttasten). Das Multibyte-Zeichen wird in *s* gespeichert. Maximal werden MB_CUR_MAX gespeichert. Hat *wc* den Wert Null, behält *wctomb* den ursprünglichen Zustand bei. Das Verhalten von *wctomb* ist von der Einstellung der LC_CTYPE-Kategorie in der aktuellen *locale*-Variablen abhängig.

Rückgabewert *wctomb* liefert ungleich Null bzw. Null zurück, wenn *s* ein Nullzeiger ist und die Multibyte-Zeichen statusabhängige Codierungen enthalten bzw. nicht enthalten.

Der Rückgabewert von *wctomb* ist -1, wenn *s* kein Nullzeiger und *wc* kein gültiges Multibyte-Zeichen ist.

In allen anderen Fällen liefert *wctomb* die Anzahl der Bytes des durch *wc* angegebenen Multibyte-Zeichens zurück.

Der Rückgabewert ist nie größer als der Wert von MB_CUR_MAX.

wherex

Funktion Liefert die Spaltenposition des Cursor innerhalb eines Fensters zurück.

Syntax
```
#include <conio.h>
int wherex(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *wherex* liefert die aktuelle Spaltenposition des Cursor zurück. Die Funktion arbeitet relativ zum aktuellen Textfenster.

Rückgabewert *wherex* liefert einen **int** im Wertebereich von 1 bis 80.

Siehe auch *gettextinfo, gotoxy, wherey*

Beispiel
```
#include <conio.h>

int main(void)
{
   clrscr();
   gotoxy(10,10);
   cprintf("Current location is X: %d  Y: %d\r\n",
           wherex(), wherey());
   getch();

   return 0;
}
```

wherey

Funktion Liefert die Zeilenposition des Cursor innerhalb eines Fensters zurück.

Syntax
```
#include <conio.h>
int wherey(void);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *wherey* liefert die aktuelle Zeilenposition des Cursor zurück. Die Funktion arbeitet relativ zum aktuellen Textfenster.

Rückgabewert *wherey* liefert einen **int** im Wertebereich von 1 bis 25, 43 oder 50 zurück.

Siehe auch *gettextinfo*, *gotoxy*, *wherex*

Beispiel
```
#include <conio.h>

int main(void)
{
   clrscr();
   gotoxy(10,10);
   cprintf("Current location is X: %d  Y: %d\r\n",
           wherex(), wherey());
   getch();

   return 0;
}
```

window

Funktion Definiert einen Bereich des Bildschirms als Textfenster.

Syntax
```
#include <conio.h>
void window(int left, int top, int right, int bottom);
```

DOS	UNIX	Windows	ANSI C	nur C++
x				

Beschreibung *window* definiert einen rechteckigen Bereich des Bildschirms als Textfenster, dessen obere linke Ecke mit (*left*, *top*) und dessen untere rechte Ecke durch (*right*, *bottom*) festgelegt ist. Die obere linke Ecke des Bildschirms hat die Koordinaten (1,1), die Minimalgröße des Fensters ist ein einzelner Buchstabe (d.h. ein Spalte und eine Zeile).

Wenn einer der vier Parameter von *window* nicht innerhalb der erlaubten Grenzen liegt, wird der Aufruf ignoriert und das aktuelle Fenster bleibt unverändert.

In den Videomodi mit 80 Spalten verwendet Borland C++ standardmäßig ein Fenster, das einem Aufruf von *window* mit den Werten (1,1,80,25) entspricht; in den Videomodi mit 40 Spalten haben diese Koordinaten die Werte (1,1,40,25).

Rückgabewert Keiner.

Siehe auch *clreol*, *clrscr*, *delline*, *gettextinfo*, *gotoxy*, *insline*, *puttext*, *textmode*

Beispiel

```
#include <conio.h>

int main(void)
{

   window(10,10,40,11);
   textcolor(BLACK);
   textbackground(WHITE);
   cprintf("This is a test\r\n");

   return 0;
}
```

_write

Funktion

Schreibt Daten in eine Datei.

Syntax

```
#include <io.h>
int _write(int handle, void *buf, unsigned len);
```

DOS	UNIX	Windows	ANSI C	nur C++
x		x		

Beschreibung

_write liest *len* Bytes ab der durch *buf* angegebenen Adresse aus dem Speicher und schreibt sie in die durch *handle* bezeichnete Datei.

Mit *_write* können maximal 65534 Bytes geschrieben werden, damit zwischen einem regulären Funktionsergebnis (das in diesem Fall als Wert vom Typ **unsigned** interpretiert werden muß) und einer Fehlermeldung (-1, entspricht 65535) unterschieden werden kann.

_write übersetzt den Zeilenvorschub ('\n') generell *nicht* in die Zeichenfolge CR/LF (0x0D, 0x0A).

Bei Diskettendateien schreibt *_write* ab der aktuellen Position (siehe *lseek*), bei Gerätedateien wird direkt zum jeweiligen Gerät geschrieben.

Wenn eine Datei im Modus »Append« (O_APPEND) geöffnet wurde, schreibt *_write* direkt ab der aktuellen Position des Dateizeigers und nicht ab dem Dateiende.

Rückgabewert

Die Funktion liefert die Anzahl der tatsächlich geschriebenen Bytes zurück. Ein Rückgabewert von weniger als *len* steht normalerweise für »Diskette voll«.

Im Falle eines Fehlers wird der Wert -1 zurückgeliefert und *errno* bekommt einen der folgenden Werte:

EACCES	Zugriff nicht gestattet
EBADF	Ungültiges Handle/Datei nicht offen

Siehe auch

lseek, *_read*, *write*

Beispiel

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>

int main(void)
{
   void *buf;
   int handle, bytes;

   buf = malloc(200);

   /*
      Create a file name TEST.$$$ in the current
      directory and writes 200 bytes to it. If
      TEST.$$$ already exists, it's overwritten.
   */

   if ((handle = open("TEST.$$$", O_CREAT | O_WRONLY |
                  O_BINARY, S_IWRITE | S_IREAD)) == -1)
   {
      printf("Error Opening File\n");
      exit(1);
   }

   if ((bytes = _write(handle, buf, 200)) == -1) {
      printf("Write Failed.\n");
      exit(1);
   }
   printf("_write: %d bytes written.\n",bytes);

   return 0;
}
```

write

Funktion Schreibt Daten in eine Datei.

Syntax

```
#include <io.h>
int write(int handle, void *buf, unsigned len);
```

DOS	UNIX	Windows	ANSI C	nur C++
x	x	x		

Beschreibung *write* liest *len* Bytes ab der durch *buf* angegebenen Adresse aus dem Speicher und schreibt sie in die durch *handle* bezeichnete

Datei. *handle* ist ein Datei-Handle, das man durch einen Aufruf von *creat*, *open*, *dup* oder *dup2* erhält.

Mit *write* sollten maximal 65534 Bytes geschrieben werden, damit zwischen einem regulären Funktionsergebnis (das in diesem Fall als Wert vom Typ **unsigned** interpretiert werden muß) und einer Fehlermeldung (-1, entspricht 65535) unterschieden werden kann.

Wenn die Datei im Modus *text* geöffnet wurde, »übersetzt« *write* jeden Zeilenvorschub ('\n') in die Zeichenfolge CR/LF (0x0D, 0x0A).

Bei Diskettendateien schreibt *write* ab der aktuellen Position (siehe *lseek*), bei Gerätedateien wird direkt zum jeweiligen Gerät geschrieben.

Wenn die Datei im Modus »Append« (O_APPEND) geöffnet wurde, setzt *write* die aktuelle Position automatisch auf das Dateiende.

Rückgabewert

Die Funktion liefert die Anzahl der tatsächlich geschriebenen Bytes zurück (wobei *write* die aus Zeilenvorschüben erzeugten CR/LF-Kombinationen als ein Byte zählt). Ein Rückgabewert von weniger als *len* steht normalerweise für »Diskette voll«.

Im Fall eines Fehlers wird der Wert -1 zurückgeliefert und *errno* bekommt einen der folgenden Werte:

EACCES	Zugriff nicht gestattet
EBADF	Ungültiges Handle

Siehe auch

creat, *lseek*, *open*, *read*, *_write*

Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>

int main(void)
{
   int handle;
   char string[40];
   int length, res;

   /*
    Create a file named "TEST.$$$" in the
    current directory and write a string to it.
    If "TEST.$$$" already exists, it will be
    overwritten.
   */

   if ((handle = open("TEST.$$$", O_WRONLY | O_CREAT |
                   O_TRUNC, S_IREAD | S_IWRITE)) == -1)
   {
```

```
      printf("Error opening file.\n");
      exit(1);
   }

   strcpy(string, "Hello, world!\n");
   length = strlen(string);

   if ((res = write(handle,string,length)) != length)
   {
      printf("Error writing to the file.\n");
      exit(1);
   }
   printf("Wrote %d bytes to the file.\n", res);

   close(handle);
   return 0;
}
```

3. Globale Variablen

Borlands C++ Compiler stellen Ihnen für viele allgemeine Standardgrößen vordefinierte globale Variablen wie Kalenderdatum und Uhrzeit, Kommandozeilenparameter, usw. zur Verfügung. Dieses Kapitel beschreibt und definiert diese globalen Variablen.

_8087

Funktion	Coprozessor-Flag.
Syntax	`extern int _8087;`
Deklaration in	dos.h
Beschreibung	Diese Variable wird durch die automatische Erkennung des Startup-Codes auf einen Wert ungleich 0 (1, 2 oder 3) gesetzt, wenn ein installierter mathematischer Coprozessor (ein 8087, 80287 oder ein 80387) erkannt wurde. Andernfalls bekommt *_8087* den Wert 0. Die automatische Erkennungslogik kann überschrieben werden, indem der *Environment*-Variablen 87 der Wert YES oder NO zugewiesen wird. Die Befehle hierzu sind: *SET 87=YES* oder *SET 87=NO*; vor und nach dem Gleichheitszeichen darf kein Leerzeichen stehen. In diesem Fall wird die globale Variable *_8087* entsprechend der *Environment*-Variablen 87 gesetzt.

_argc

Funktion	Zahl der Kommandozeilenparameter.
Syntax	`extern int _argc;`
Deklaration in	dos.h
Beschreibung	*_argc* enthält die als *argc* an *main* übergebene Anzahl der Kommandozeilenparameter.

_argv

Funktion	Ein Array mit Zeigern auf die einzelnen Kommandozeilenparameter.
Syntax	`extern char *_argv[];`
Deklaration in	dos.h
Beschreibung	*_argv* zeigt auf ein String-Array mit den einzelnen Kommandozeilenparametern, die *main* beim Programmstart als Elemente von *argv*[] übergeben wurden.

_ctype

Funktion	Ein Array mit Zeichenattribut-Informationen.
Syntax	`extern char _ctype[];`
Deklaration in	ctype.h
Beschreibung	*_ctype* ist ein Array mit Zeichenattribut-Informationen. Der Index ist der ASCII-Wert +1. Jeder Eintrag ist ein Satz von Bits, die das Zeichen beschreiben. Dieses Array wird von *isdigit*, *isprint* usw. verwendet.

daylight

Funktion	Legt fest, ob Anpassungen an die Sommerzeit durchgeführt werden.
Syntax	`extern int daylight;`
Deklaration in	time.h
Beschreibung	Die Variable wird zusammen mit den Funktionen für Uhrzeit und Datum verwendet. *daylight* muß von *tzset*, *ftime* oder *localtime* auf den Wert 1 gesetzt werden, wenn die Sommerzeit berücksichtigt werden soll; 0 steht für die Standardzeit.

directvideo

Funktion	Flag für die Steuerung von Bildschirmausgaben.
Syntax	`extern int directvideo;`
Deklaration in	conio.h
Beschreibung	*directvideo* steuert, ob Bildschirmausgaben Ihres Programms (z.B. von *cputs*) direkt in den Bildschirmspeicher schreiben (*directvideo* = 1) oder BIOS-Funktionen verwenden (*directvideo* = 0). Die Voreinstellung ist *directvideo* = 1 (die Ausgabe schreibt

in den Bildschirmspeicher) und sollte nur verwendet werden, wenn die Video-Hardware Ihres Systems 100% kompatibel zu einer IBM-Grafikkarte ist. Mit *directvideo* =0 benötigen Sie nur ein System, das IBM-BIOS-kompatibel ist.

environ

Funktion: Zugriff auf die DOS-Environment-Variablen.

Syntax:
```
extern char * environ[];
```

Deklaration in: dos.h

Beschreibung: Die Variable *environ* ist ein Array von Zeigern auf Strings, über die die aktuellen Einträge der DOS-Tabelle *Environment* ermittelt und geändert werden können. Jeder Eintrag dieser Tabelle hat die folgende Form:

```
envvar = varvalue
```

wobei *envvar* der Name einer Environment-Variablen (wie z.B. PATH) und *varvalue* der Stringwert (wie z.B. C:\BIN;C:\DOS) ist, der *envvar* zugewiesen wird. *varvalue* kann auch leer sein.

Beim Start eines Programms werden die Einstellungen des DOS-Environment direkt an das Programm übergeben. Der Inhalt von *env*, dem dritten Parameter von *main*, stimmt mit dem Anfangswert von *environ* überein.

Auf *environ* kann auch über *getenv* zugegriffen werden; um Einträge der *Environment*-Tabelle zu ändern, löschen oder hinzuzufügen, sollte jedoch ausschließlich *putenv* verwendet werden. Modifikationen können die Größe und damit den Speicherbedarf des Environment verändern, *environ* aber wird automatisch angepaßt und zeigt somit immer auf das *Environment*-Array.

Siehe auch: *getenv*, *putenv*

errno, _doserrno, sys_errlist, sys_nerr

Funktion: Ermöglicht *perror* die Ausgabe von Fehlermeldungen.

Syntax:
```
extern int errno;
extern int _doserrno;
extern char *sys_errlist[];
extern int sys_nerr;
```

Deklaration in: errno.h, stdlib.h: *errno*, *_doserrno*; *sys_errlist*, *sys_nerr*
dos.h: *_doserrno*

Beschreibung: *errno*, *sys_errlist* und *sys_nerr* werden von *perror* zur Ausgabe von Fehlermeldungen benutzt, wenn bestimmte Bibliotheksfunktionen ihre Aufgabe nicht ausführen können. Die Variable

_doserrno bildet viele DOS-Fehlercodes auf *errno* ab; *_doserrno* wird von *perror* aber nicht direkt benutzt.

_doserrno: Wenn ein DOS-Systemaufruf mit einem Fehler endet, wird der aktuelle Fehlercode von DOS in *_doserrno* gespeichert. Die zusätzlich und gleichzeitig mitgeführte Fehlervariable *errno* stammt aus dem UNIX-Umfeld.

errno: Bei einem Fehler nach einem Aufruf einer mathematischen Funktion oder einer Systemfunktion wird *errno* gesetzt, um den Fehlertyp anzuzeigen. Manchmal sind *errno* und *_doserrno* äquivalent, in anderen Fällen enthält *errno*, anders als *_doserrno*, nicht den aktuellen DOS-Fehlercode. Bestimmte Fehler setzen nur *errno*, nicht aber *_doserrno*.

sys_errlist: Das Array mit den Strings, die die Fehlermeldungen enthalten, steht Ihnen in der Variablen *sys_erlist* zur Verfügung und ermöglicht Ihnen so bessere Steuerung der Formatierung der Meldungen. Benutzen Sie *errno* als Index in dieses Array, um den zur aktuellen Fehlernummer gehörigen String aufzufinden. Diese Strings enthalten keine Zeilenvorschubzeichen.

sys_nerr: Diese Variable enthält die Anzahl der Fehlermeldungsstrings in *sys_errlist*.

Folgende Symbole und Fehlermeldungen sind in errno.h (bzw. in *sys_errlist*) definiert:

E2BIG	*Arg list too long* (Argumentenliste zu lang)
EACCES	*Permission denied* (Zugriff nicht gestattet)
EBADF	*Bad file number* (Ungültige Dateinummer/Ungültiges Handle)
ECONTR	*Memory blocks destroyed* (Speicherbereiche zerstört)
ECURDIR	*Attempted to remove CurDir* (Versuch, das aktuelle Verzeichnis zu löschen)
EDOM	*Domain error* (Ungültiges Argument für mathematische Funktion/Bereichsfehler)
EEXIST	*File already exists* (Datei existiert bereits)
EFAULT	*Unknown error* (Unbekannter Fehler)
EINVACC	*Invalid access code* (Ungültiger Zugriffscode)
EINVAL	*Invalid argument* (Ungültiges Argument)
EINVDAT	*Invalid data* (Ungültige Daten)
EINVDRV	*Invalid drive specified* (Ungültiges Laufwerk angegeben)

EINVENV	*Invalid environment* (Ungültiges *Environment*)
EINVFMT	*Invalid format* (Ungültiges Format)
EINVFNC	*Invalid function number* (Ungültige Funktionsnummer)
EINVMEM	*Invalid memory block address* (Ungültige Speicherblockadresse)
EMFILE	*Too many open files* (Zu viele Dateien bereits offen)
ENMFILE	*No more files* (Keine weiteren Dateieinträge)
ENODEV	*No such device* (Angegebenes Gerät existiert nicht)
ENOENT	*No such file or directory* (Datei/Suchweg nicht gefunden)
ENOEXEC	*Exec format error* (exec-Formatfehler)
ENOFILE	*No such file or directory* (Datei/Suchweg nicht gefunden)
ENOMEM	*Not enough memory* (Nicht genug Platz im Hauptspeicher)
ENOPATH	*Path not found* (Suchweg nicht gefunden)
ENOTSAM	*Not same device* (Ziel- und Quellgerät sind nicht identisch)
ERANGE	*Result out of range* (Ergebnis außerhalb des Rechenbereichs)
EXDEV	*Cross-device link* (»Rename« mit verschiedenen Laufwerken)
EZERO	*Error 0* (»Kein Fehler«)

Für direkt von DOS zurückgelieferte Fehler (d.h. mögliche Werte von *_doserrno*) sind die folgenden Konstanten in errno.h definiert, deren Werte und Namen sich teilweise mit denen von *errno* überschneiden:

E2BIG	*Bad environ* (Ungültiges Environment)
EACCES	*Access denied* (Zugriff verweigert)
EACCES	*Bad access* (Falscher Zugriff)
EACCES	*Is current dir* (Aktuelles Verzeichnis)
EBADF	*Bad handle* (Ungültiges Handle)
EFAULT	*Reserved*

EINVAL	*Bad data* (Ungültige Daten)
EINVAL	*Bad function* (Ungültige Funktion)
EMFILE	*Too many open* (Zu viele Dateien geöffnet)
ENOENT	*No such file or directory* (Datei/Suchweg nicht gefunden)
ENOEXEC	*Bad format* (Ungültiges Format)
ENOMEM	*Mcb destroyed* (MCB zerstört)
ENOMEM	*Out of memory* (Nicht ausreichend Speicher)
ENOMEM	*Bad block* (Ungültiger Block)
EXDEV	*Bad drive* (Ungültiges Laufwerk)
EXDEV	*Not same device* (Ziel- und Quellgerät sind nicht identisch)

Eine detailliertere Beschreibung der Fehlerwerte von DOS finden Sie im DOS-Referenzhandbuch.

Beispiel

```
#include <errno.h>
#include <stdio.h>

extern char *sys_errlist[];

main()
{
   int i = 0;

   while(sys_errlist[i++]) printf("%s\n",
         sys_errlist[i]);
   return 0;
}
```

_fmode

Funktion — Legt *text* oder *binary* für die Eröffnung von Dateien fest.

Syntax

```
extern int _fmode;
```

Deklaration in — fcntl.h

Beschreibung — *_fmode* legt fest, in welchem Modus (*text* oder *binary*) Dateien geöffnet und übersetzt werden. Die Voreinstellung für *_fmode* ist O_TEXT, wodurch Dateien im Textmodus gelesen werden. Um Dateien im Binärmodus zu öffnen und zu lesen, muß *_fmode* auf O_BINARY gesetzt werden. Die Konstanten O_TEXT und O_BINARY sind in fcntl.h definiert.

Im Textmodus werden bei der Eingabe CR/LF-Kombinationen (Wagenrücklauf/Zeilenvorschub) in ein einzelnes LF-Zeichen übersetzt. Bei der Ausgabe werden umgekehrt einzelne LF-Zeichen in CR/LF-Kombinationen übersetzt.

Im Binärmodus findet keinerlei Übersetzung statt.

Der durch *_fmode* voreingestellte Modus kann durch Angabe eines *t* (für Textmodus) oder *b* (für Binärmodus) als Parameter *type* bei den Bibliotheksfunktionen *fopen*, *fdopen* und *freopen* überschrieben werden. Auch in der Funktion *open* kann durch Angabe einer der beiden Konstanten O_BINARY oder O_TEXT im Parameter *access* Binärmodus bzw. Textmodus für die zu öffnende Datei explizit gewählt werden.

_heaplen

Funktion Enthält die Größe des Near-Heap.

Syntax
```
extern unsigned _heaplen;
```

Deklaration in dos.h

Beschreibung *_heaplen* enthält die Größe des Near-Heap in Bytes und existiert nur in den kleinen Speichermodellen (*tiny*, *small* und *medium*); in den größeren Speichermodellen (*compact*, *large* und *huge*) gibt es keinen Near-Heap.

In den Modellen *small* und *medium* wird die Größe des Datensegments wie folgt berechnet:

```
Datensegment [small, medium] = globale Daten + Heap
                             +   Stack
```

Die Größe des Stack kann mit *_stklen* angepaßt werden.

Wenn *_heaplen* auf 0 gesetzt wird, reserviert das Programm 64 KByte des Speichers für das Datensegment, die effektive Größe des Heap beträgt dann:

```
64 KByte - (Globale Daten + Stack) Bytes
```

Die Standardgröße für *_heaplen* ist 0. Wird daher kein Wert für *_heaplen* angegeben, erhält man ein 64 KByte großes Datensegment.

Im Modell *tiny* befindet sich alles (auch der Programmcode) in demselben Segment, so daß die Größe des Programmcodes plus 256 Bytes für das Programmsegmentpräfix (PSP) für die Berechnung der Datensegmentgröße berücksichtigt wird:

```
Datensegment [tiny] = 256 Bytes PSP + Programmcode
                    + globale Daten + Heap
                    + Stack
```

Wenn *_heaplen* im Modell *tiny* auf 0 gesetzt wird, berechnet sich die sich die effektive Größe des Heap wie folgt:

```
Heapgröße = 64 KByte - (Programmcode + 256 Bytes PSP
                        globale Daten + Stack)
```

In den Speichermodellen *compact* und *large* gibt es keinen Near-Heap. Der Stack befindet sich in einem eigenen Segment, die Größe des Datensegments ist:

```
Datensegment [compact, large] = globale Daten
```

Im Speichermodell *huge* ist der Stack ein separates Segment, jedes Modul hat hier sein eigenenes Datensegment.

Siehe auch *_stklen*

_new_handler

Funktion Fängt Fehler bei der Reservierung von neuem Speicherplatz ab.

Syntax
```
typedef void(*pvf)();
pvf _new_handler
```

Statt dieser direkten Zuweisung ist auch eine alternative Formulierung über den Aufruf der Funktion *_set_new_handler* möglich:

```
pvf set_new_handler(pvf p);
```

Beschreibung *_new_handler* beeinhaltet einen Zeiger auf eine Funktion, der keine Parameter übergeben werden und die einen Rückgabewert vom Typ **void** hat. Wenn *operator new*() den benötigten Speicherplatz nicht reservieren kann, wird die Funktion aufgerufen, auf die *_new_handler* zeigt. Nach der Rückkehr von dieser Funktion wird erneut versucht, den Speicherplatz zu reservieren. Per Voreinstellung weist *_new_handler* auf eine Funktion, die das Programm beendet. Das Programm kann diesen Handler durch eine Funktion ersetzen, die versucht, Speicherplatz freizugeben. Dies kann über eine direkte Zuweisung auf *_new_handler* oder durch den Aufruf der Funktion *_set_new_handler* geschehen. *set_new_handler* liefert einen Zeiger auf den früheren Handler zurück.

_new_handler wird primär aus Kompatibilitätsgründen zur C++Version 1.2 von AT&T bereitgestellt. In den meisten Fällen erreicht man diese Funktionalität besser durch Überladen von *operator new*().

_osmajor, _osminor

Funktion Haupt- und Neben-Versionsnummer des Betriebssystems.

Syntax
```
extern unsigned char _osmajor;
extern unsigned char _osminor;
```

Deklaration in dos.h

Beschreibung Die Haupt- und Neben-Versionsnummern von DOS stehen über *_osmajor* und *_osminor* zur Verfügung. *_osmajor* beinhaltet die Haupt-Versionsnummer, *_osminor* die Neben-Versionsnummer von DOS. Z.B. ist unter der DOS-Version 3.2 der Wert von *_osmajor* auf 3, der Wert von *_osminor* auf 20 gesetzt.

Die Variablen sind sehr hilfreich, wenn ein Programm unter DOS-Versionen 2.x und 3.x laufen soll. Einige Bibliotheksfunktionen verhalten sich in Abhängigkeit von der DOS-Versionsnummer unterschiedlich, andere funktionieren nur unter DOS 3.x. Vergleichen Sie hierzu die Beschreibungen von *open*, *creatnew* und *ioctl* in Kapitel 2 »Die Bibliotheksfunktionen« dieses Handbuches.

_ovrbuffer

Funktion Ändert die Größe des Overlay-Puffers.

Syntax
```
unsigned _ovrbuffer = size
```

Deklaration dos.h

Beschreibung Die Standardgröße des Overlay-Puffers entspricht dem doppelten Umfang des größten Overlay. Das ist für einige Anwendungen sicher angemessen.

Wenn jedoch eine bestimmte Funktion eines Programms mit Hilfe vieler Overlay-Module implementiert wurde, dann kann es Probleme geben: Sobald die Gesamtgröße aller Module größer als der Overlay-Puffer ist, werden Module ein- und ausgelagert. Dies ist umso häufiger der Fall, wenn sich die Module bei der Ausführung der Funktion oft wechselseitig aufrufen müssen.

Zur Lösung dieses Problems kann die Größe des Overlay-Puffers erhöht werden, so daß jederzeit genügend Speicherplatz für Overlay-Module zur Verfügung steht, die sich häufig wechselseitig aufrufen. Dazu wird die globale Variable *_ovrbuffer* auf die benötigte Größe gesetzt.

Dazu ein Beispiel: Die folgende Anweisung setzt den Overlay-Puffer auf 128 KByte:

```
unsigned _ovrbuffer = 0x2000;
```

Es gibt keine Formel für die Berechnung der idealen Größe des Overlay-Puffers. Unterstützung bei der praktischen Ermittlung einer angemessenen Puffergröße bietet der Turbo Profiler von Borland.

Siehe auch *_OvrInitEms*, *_OvrInitExt*

_psp

Funktion Enthält die Segmentadresse des Programmsegmentpräfixes (PSP) des aktuellen Programms.

Syntax
```
extern unsigned int _psp;
```

Deklaration in dos.h

Beschreibung Die Variable *_psp* enthält die (normalisierte) Segment-Adresse des Programmsegmentpräfix beim Start eines Programms – eines von DOS reservierten Bereichs, der interne Informationen wie die höchste verfügbare Speicheradresse, Kommandozeilenparameter und anderes enthält. Ausführliche Informationen zum PSP finden Sie in Ihrem DOS-Referenzhandbuch.

_stklen

Funktion Enthält die Größe des Stack.

Syntax
```
extern unsigned _stklen;
```

Deklaration in dos.h

Beschreibung *_stklen* enthält die Größe des Stack in allen sechs Speichermodellen. Die Minimalgröße von *_stklen* beträgt 256 Bytes (128 WORD); kleinere Werte werden automatisch durch dieses Minimum ersetzt. Die Voreinstellung für die Größe des Stack ist 4 KByte.

In den Modellen *small* und *medium* wird die Größe des Datensegments wie folgt berechnet:

```
Datensegment [small, medium] = globale Daten + Heap
                             +   Stack
```

Die Größe des Heap kann mit *_heaplen* angepaßt werden.

Im Modell *tiny* befindet sich alles (auch der Programmcode) in demselben Segment, so daß die Größe des Programmcodes plus 256 Bytes für das Programmsegmentpräfix (PSP) für die Berechnung der Datensegmentgröße berücksichtigt wird:

```
Datensegment [tiny] = 256 Bytes PSP + Programmcode
                    + globale Daten + near Heap
                    + Stack
```

In den Speichermodellen *compact* und *large* gibt es keinen Near-Heap, der Stack befindet sich hier in einem eigenen Segment.

Die Größe des Datensegment ist dann einfach:

```
Datensegment [compact, large] = globale Daten
```

Im Modell *huge* ist der Stack ein separates Segment, jedes Modul hat hier sein eigenenes Datensegment.

Siehe auch `_heaplen`

Beispiel

```
#include <stdio.h>
/* Set the stack size to be greater than the default. */
/* This declaration must go in the global data area. */

extern unsigned _stklen = 54321U;

main()
{
   /* show the current stack length */
   printf("The stack length is: %u\n", _stklen);
   return 0;
}
```

timezone

Funktion Enthält den Unterschied zwischen der lokalen Zeit und der GMT in Sekunden.

Syntax `extern long timezone;`

Deklaration in time.h

Beschreibung Die Variable wird zusammen mit den Funktionen für Uhrzeit und Datum verwendet. *timezone* wird von der Funktion *tzset* berechnet. *timezone* ist ein **long**-Wert, der den Unterschied zwischen der Greenwich Mean Time und der lokalen Zeit in Sekunden wiedergibt.

tzname

Funktion Ein Array von Zeigern auf die Namen der Zeitzonen.

Syntax `extern char *tzname[2];`

Deklaration in time.h

Beschreibung Die globale Variable *tzname* ist ein Array von Zeigern auf Strings, die Abkürzungen von Zeitzonennamen enthalten. *tzname*[*0*] zeigt auf einen String mit drei Zeichen, der den Wert des Zeitzonennamens aus der *Environment*-Variablen *TZ* enthält. *tzname*[*1*] zeigt auf einen String mit drei Zeichen, der den Wert des Sommerzeitnamens aus der *Environment*-Variablen *TZ* enthält.

Wenn die Konvertierung in die Sommerzeit nicht benutzt wird, zeigt *tzname*[*1*] auf einen Nullstring.

_version

Funktion	Versionsnummer von DOS.
Syntax	`extern unsigned int _version;`
Deklaration in	dos.h
Beschreibung	*_version* enthält die Nummer der DOS-Version, die Haupt-Versionsnummer im niederwertigen Byte und die Neben-Versionsnummer im höherwertigen Byte. Bei der DOS-Version *x.y* ist *x* die Haupt- und *y* die Neben-Versionsnummer.

_wscroll

Funktion	Ein- oder Abschalten des Bildlaufs bei Bildschirm-Ein-/Ausgabefunktionen.
Syntax	`extern int _wscroll;`
Deklaration in	conio.h
Beschreibung	Die Variable *_wscroll* ist ein Flag für Bildschirm-Ein-/Ausgabefunktionen. Die Voreinstellung ist 1. Setzen Sie *_wscroll* auf 0, so ist der Bildlauf ausgeschaltet. Damit verhindern Sie unerwünschten Bildlauf, wenn Sie Zeichen in der unteren rechten Ecke eines Fensters ausgeben.

Index

0x11, BIOS-Interrupt 45f, 48
0x12, BIOS-Interrupt 52f
0x13, BIOS-Interrupt 42
0x16, BIOS-Interrupt 48, 50
0x17, BIOS-Interrupt 53, 55
0x19, DOS-Funktion 123, 232f
0x1A, BIOS-Interrupt 59f
0x21, DOS-Interrupt 324f
0x23, DOS-Interrupt 107, 290, 297
0x24, DOS-Interrupt 290, 294
0x25, DOS-Interrupt 12
0x26, DOS-Interrupt 13
0x27, DOS-Funktionsaufruf 439
0x28, DOS-Funktionsaufruf 441
0x31, DOS-Funktion 340
0x33, DOS-Systemaufruf 221, 490
0x38, DOS-Funktion 298
0x3F, DOS-Funktion 445
0x40, DOS-Funktion 130
0x44, DOS-Systemaufruf 327
0x48, DOS-Systemaufruf 17
0x4E, DOS-Systemaufruf 119, 179
0x59, DOS-Systemaufruf 118
0x5B, DOS-Funktion 117
0x62, DOS-Systemaufruf 264
8086-Prozessor
  Interrupts 321f, 326
  Interrupt-Vektoren 127f, 272
80x87
  siehe mathematischer Koprozessor
_8087 (globale Variable) 633

A
Abbruch
  von Programmen 11, 148f
  von Programmen mit assert 23
abort (Bibliotheksfunktion) 11
abrunden 184
abs (Bibliotheksmakro) 12
absolute Werte 151, 344
  Integer 11
  komplexe Zahlen und 64, 392
  von Gleitkommazahlen 151
absread (Bibliotheksfunktion) 12
abswrite (Bibliotheksfunktion) 13
access (Bibliotheksfunktion) 14
acos (Bibliotheksfunktion) 15
Adapter
  Grafik-
  siehe Grafikadapter
Adreßsegment
  far-Zeiger 191
Adressen
  an __emit__ übergebene 139
  far 191
  siehe auch Speicher
aktive Grafikseite 479
aktuelle Position (CP) 349
  bewegen 388
  Koordinaten 524
  Position ermitteln 277f
  setzen 390
alloca (Bibliotheksfunktion) 16
allocmem (Bibliotheksfunktion) 17
alphabetische Zeichen
  prüfen auf 330
alphanumerische Zeichen
  prüfen auf 329
arc (Bibliotheksfunktion) 19
  Koordinaten ermitteln 216
arccoordstype (Datentyp) 216
Arcuscosinus 15
Arcussinus 22
Arcustangens 24f

arg (Bibliotheksfunktion) 20, 91
_argc (globale Variable) 633
argc
  Parameter für main 3
ARGS.EXE 4
_argv (globale Variable) 634
argv
  Parameter für main 3
Arrays
  absuchen 62, 346
  Elemente sortieren 436
  Zeichenattribute 634
ASCII-Format
  Zeichen kovertieren in 600
ASCII-Zeichen
  prüfen 329ff
asctime (Bibliotheksfunktion) 21
asin (Bibliotheksfunktion) 22
asinl (Bibliotheksfunktion) 22
assert (Bibliotheksmakro) 24
atan (Bibliotheksfunktion) 24
atan2 (Bibliotheksfunktion) 25
atan2l (Bibliotheksfunktion) 25
atanl (Bibliotheksfunktion) 24
atexit (Bibliotheksfunktion) 26
atof (Bibliotheksfunktion) 27
atoi (Bibliotheksfunktion) 28
atol (Bibliotheksfunktion) 29
_atold (Bibliotheksfunktion) 27
Attribute 75
  Text 588, 590f
  Zeichen 634
Aufruf
  DOS-Kommando 583
Ausgabe
  Fehlermeldungen 635
  formatiert zu stdout 418
  formatiert zu Streams 194
  formatierte 97, 418, 545, 616f, 619, 622
  Pufferung 522
  von Strings 98, 196, 433
  siehe auch Ein-/Ausgabe
Ausrichtung
  von Textausgaben 515

B

Balkengrafiken 30, 32
bar (Bibliotheksfunktion) 30
bar3d (Bibliotheksfunktion) 32
Baudrate 37, 56
bcd (Bibliotheksfunktion) 33
BCD-Zahlen
  konvertieren 450
bdos (Bibliotheksfunktion) 34
bdosptr (Bibliotheksfunktion) 35
Bedingung prüfen 23
Behandlungsroutinen 107
Berechnungsfehler
  Behandlung 370
BGI
  Grafiktreiber und 288, 310, 317
Bibliotheksfunktionen 9ff
Bildausschnitte
  Ausgabe auf Bildschirm 429
  im Textmodus 266, 434
  in der Grafik 245, 308
  kopieren 389
Bildlauf
  _wscroll (glob. Variable) und 644
Bildpunkte
  ermitteln 263
  setzen 432
Bildschirm
  Ausgabe auf 223, 426, 634
  Ausschnitt speichern 245, 308
  direktes Schreiben 635
  formatierte Ausgabe auf 97
  Höhen-/Seitenverhältnis 218, 484
  löschen 81, 90, 501
  maximale Koordinaten 253f
  Strings anzeigen 98
  Text speichern 266
Bildschirmausgabe-Routinen
  gettext 266
  gettextinfo 267
  gotoxy 280
  highvideo 305
  insline 316
  lowvideo 360
  movetext 389
  normvideo 393
  puttext 434
  textattr 588
  textbackground 590
  textcolor 591
  textmode 594
  wherex 626
  wherey 626
  window 627
Bildschirmmodus

wiederherstellen 458
binär
Modus 130, 397, 537, 628, 630
Suche 62
binäre Dateien 599
_fsopen und 206
creat und 101
creattemp und 104
fdopen und 167
öffnen 638
temporär öffnen 598
BIOS-Timer 59f
BIOS-Interrupts
0x11 45ff
0x12 52f
0x13 42
0x16 48, 50
0x17 53, 55
0x1A 59f
bioscom (Bibliotheksfunktion) 36
biosdisk (Bibliotheksfunktion) 42
_bios_disk (Bibliotheksfunktion) 39
biosequip (Bibliotheksfunktion) 45
_bios_equiplist (Bibliotheksfunktion) 46
bioskey (Bibliotheksfunktion) 48
_bios_keybrd (Bibliotheksfunktion) 50
biosmemory (Bibliotheksfunktion) 52
_bios_memsize (Bibliotheksfunktion) 53
biosprint (Bibliotheksfunktion) 53
_bios_printer (Bibliotheksfunktion) 55
_bios_serialcom (Bibliotheksfunktion) 56
biostime (Bibliotheksfunktion) 59
_bios_timeofday (Bibliotheksfunktion) 60
bitweise Rotation 361f, 463f
block-orientierte Dateien 490, 522
Blockoperationen
siehe Editieren
Break-Wert 61, 465
Breite von Textausgaben 596
brk (Bibliotheksfunktion) 61
bsearch (Bibliotheksfunktion) 62
Bytes
Ausgabe zu I/O-Adresse 400
lesen aus Schnittstelle 315
lesen aus Speicher 410
Positionen vertauschen 583
speichern 414
Zeichen aus mehreren 373f

C

C++
binär codierte Dezimalzahl 33
Speicherbelegung 640
cabs (Bibliotheksfunktion) 64
cabsl (Bibliotheksfunktion) 64
calloc (Bibliotheksfunktion) 66
Carry-Flag 322, 324f
ceil (Bibliotheksfunktion) 67
ceill (Bibliotheksfunktion) 67
_c_exit (Bibliotheksfunktion) 68
_cexit (Bibliotheksfunktion) 69
CGA 19
cgets (Bibliotheksfunktion) 70
_chain_intr (Bibliotheksfunktion) 71
chdir (Bibliotheksfunktion) 73
_chdrive (Bibliotheksfunktion) 74
child-Prozeß 142, 540
.CHR-Dateien 517
_chmod (Bibliotheksfunktion) 75
chmod (Bibliotheksfunktion) 77
circle (Bibliotheksfunktion) 79
clear87_ (Bibliotheksfunktion) 80
cleardevice (Bibliotheksfunktion) 81
clearerr (Bibliotheksfunktion) 83
clearviewport (Bibliotheksfunktion) 84
CLK_TCK (Konstante) 85
clock (Bibliotheksfunktion) 85
clock_t (Datentyp) 85
_close (Bibliotheksfunktion) 86
close (Bibliotheksfunktion) 86
closedir (Bibliotheksfunktion) 87
closegraph (Bibliotheksfunktion) 88
clreol (Bibliotheksfunktion) 89
clrscr (Bibliotheksfunktion) 90
COMMAND.COM 583
COMSPEC (Environment-Variable) 583
conj (Bibliotheksfunktion) 92
_control87 (Bibliotheksfunktion) 92
coreleft (Bibliotheksfunktion) 93
cos (Bibliotheksfunktion) 94
cosh (Bibliotheksfunktion) 95
coshl (Bibliotheksfunktion) 95
Cosinus 15, 94
cosl (Bibliotheksfunktion) 94
country (Bibliotheksfunktion) 96
cprintf (Bibliotheksfunktion) 97
cputs (Bibliotheksfunktion) 98
CR/LF-Zeichenfolge 130, 628, 630
_creat (Bibliotheksfunktion) 99

creat (Bibliotheksfunktion) 101
creatnew (Bibliotheksfunktion) 102
creattemp (Bibliotheksfunktion) 104
cscanf (Bibliotheksfunktion) 105
ctime (Bibliotheksfunktion) 106
ctrlbrk (Bibliotheksfunktion) 107
_ctype (globale Variable) 634
Cursor
  Form festlegen 493
  Position ermitteln 626
  Position im Textfenster 280
  Position setzen 280

D
Datei-Handle 116
  Stream zuordnen 166
  verdoppeln 134f
Dateiattribute 75, 77, 99ff, 104, 119
  DOS 124
  setzen 117
Dateien
  suchen 119, 121, 179ff
  Suchwege analysieren/konstruieren 186, 188, 367, 543
  ARGS.EXE 4
  Attribute 75, 77, 99ff, 104, 119
  binäre 167, 397, 509, 537
  .CHR 320
  Dateiende-Bestimmung 141, 169
  Datum und Uhrzeit 125, 243, 613
  eindeutige Namen 587
  erzeugen 99, 101f, 104, 117
  FCB (file control block) 439, 441
  formatierte Ausgabe 194, 418, 545, 616ff
  formatierte Eingabe 105, 466, 547, 618ff
  Größe ändern 78
  Handle 134f, 166
  Informationen über 208
  Längenbestimmung 174
  lesen 101, 444, 447
  lesen aus Streams 173
  lesen und formatieren 202, 466, 618, 620
  Lesen/Schreiben-Zugriff 101, 197
  Löschen 456, 611
  Modus 101f, 117, 124, 167, 190, 200
  Namensanalyse 407
  neue 104
  offene 208
  öffnen 190, 394ff, 399, 536
  öffnen zur Aktualisierung 167
  Position ermitteln 172
  Positionierung 210, 365, 459, 586
  Referenznummer 175
  schließen 86, 116, 163f, 200
  schreiben in 130, 213, 628f
  SHARED-Modus 206, 353, 355, 395, 536, 612
  Sperre aufheben 612
  sperren 353, 355
  Streams öffnen 190
  suchen 475f
  temporäre 587
  temporäre benennen 599
  temporäre löschen 462
  temporäre öffnen 598
  Textmodus 398, 537
  überschreiben 99, 101
  umbenennen 457
  Verzeichnis absuchen 179
  WILDARGS.OBJ 5
  Zeichen lesen aus 171, 221
  Zeiger
  siehe Dateizeiger
  Zugriffsmodus 14, 75, 77, 124, 167, 190, 200, 206, 208, 606
Dateiende-Flag 83, 141, 447
Dateimodus 101f, 117, 124, 167, 190, 200
  festlegen 638
  setzen 509
Dateinamen 205
  eindeutige 599
  eindeutige erzeugen 384
Dateizeiger 172, 445, 447
  aktuelle Position ermitteln 586
  aktuelle Position setzen 365
  initialisieren 459
  setzen 203, 205, 395, 538
  zurückliefern 210
Daten
  kopieren 387
  lesen aus Streams 202, 618, 620
Datenbits 37, 56
Datensegment 66, 388
  Größe ändern 465
  Speicherbelegung 61
Datensicherheit 261
Datentypen
  Bibliothek 64, 243
  clock_t 85
  fpos_t 173

sig_atomic_t 530
struct arccoordstype 216
struct COUNTRY 96
struct date 228
struct dfree 231
struct diskfree_t 122
struct div_t 115
struct dosdate_t 228
struct DOSERROR 118
struct dostime_t 126
struct exception 370
struct fatinfo 237f
struct ffblk 179
struct fillsettingstype 240
struct find_t 119
struct linesettingstype 248, 505
struct palettetype 259, 481
struct REGPACK 326
struct stat 208
struct text_info 267
struct textsettingstype 269
struct time 271
struct timeb 211
struct tm 279, 352
struct viewporttype 274
va_list 615
Datum und Uhrzeit
BIOS-Timer 59f
Datei 243, 613
daylight, timezone (Variablen) 604, 634
globale Variablen 604ff
GMT 279
konvertieren 352
konvertieren in Kalenderformat 385
konvertieren in String 558
konvertieren in UNIX-Format 131
landesspezifische Formate 96
lesen und setzen 125, 271
MEZ 106
Sommerzeit 634
System 21, 106, 126, 131, 211, 228, 352, 550, 597
UNIX- zu DOS-Format 610
verbrauchte CPU-Zeit seit Aufruf 85
daylight (globale Variable) 604, 634
delay (Bibliotheksfunktion) 108
delline (Bibliotheksfunktion) 109
detectgraph (Bibliotheksfunktion) 109
Diagnoseroutinen
assert 23
_matherrl 370
matherr 370
perror 411
difftime (Bibliotheksfunktion) 112
directvideo (globale Variable) 634
Disketten
freien Platz ermitteln 122, 231
Sektorenüberprüfung 523
Diskettenroutinen
absread 12
abswrite 13
_bios_disk 39
biosdisk 42
chdir 73
_chdrive 74
_dos_getdiskfree 122
getcurdir 226
getcwd 227
_getdcwd 229
getdfree 231
getdisk 232
getdta 235
getfat 236
getverify 273
setdisk 232
setdta 494
setverify 523
Diskettensektoren
lesen 12, 40, 42
schreiben 13, 40, 42
div (Bibliotheksfunktion) 115
div_t (Datentyp) 115
Division
Integer 115, 186, 345
Divisionsrest 115, 186, 345
DMA (Direct memory access) 44ff
_dos_allocmem (Bibliotheksfunktion) 17
_dos_close (Bibliotheksfunktion) 116
_dos_creat (Bibliotheksfunktion) 99
_dos_creatnew (Bibliotheksfunktion) 117
_dos_findfirst (Bibliotheksfunktion) 119
_dos_findnext (Bibliotheksfunktion) 121
_dos_freemem (Bibliotheksfunktion) 198
_dos_getdate (Bibliotheksfunktion) 228
_dos_getdiskfree (Bibliotheksfunktion) 122
_dos_getdrive (Bibliotheksfunktion) 123
_dos_getfileattr (Bibliotheksfunktion) 124
_dos_getftime (Bibliotheksfunktion) 125
_dos_gettime (Bibliotheksfunktion) 126

_dos_getvect (Bibliotheksfunktion) 127
_dos_keep (Bibliotheksfunktion) 340
_dos_open (Bibliotheksfunktion) 394
_dos_read (Bibliotheksfunktion) 444
_dos_setblock (Bibliotheksfunktion) 487
_dos_setdrive (Bibliotheksfunktion) 123
_dos_setfileattr (Bibliotheksfunktion) 124
_dos_setftime (Bibliotheksfunktion) 125
_dos_settime (Bibliotheksfunktion) 126
_dos_setvect (Bibliotheksfunktion) 128
_dos_write (Bibliotheksfunktion) 130
_doserrno (globale Variable) 635
DOS
  Attribut-WORD 117
  Dateiattribute 75, 99, 104, 117, 124
  Datum und Uhrzeit 228, 601
  Environment-Variablen 143, 541, 635
  erweiterter Fehlercode 118
  Fehlernummer (Konstanten) 637
  Pfad durchsuchen 475
  Programmsegmentpräfix 642
  Verify-Flag 273, 523
  Versionsnummer 640, 644
  Verzeichnis löschen 461
DOS-Funktionen
  0x19 123, 232f
  0x31 340
  0x38 298
  0x3F 445
  0x40 130
  0x4E 119
  0x5B 117
DOS-Interrupts
  0x21 324f
  0x23 107, 290, 297
  0x24 290, 294
  0x25 12
  0x26 13
  Funktionen 272
DOS-Kommandos
  aufrufen 583
DOS-Systemaufrufe 35
  0x27 439
  0x28 441
  0x33 221, 490
  0x44 327
  0x48 17
  0x4E 179
  0x59 118
  0x62 264
  Speichermodelle und 34f
DOS/BIOS-Routinen
  _bios_disk 39
  _bios_equiplist 46
  _bios_keybrd 50
  _bios_memsize 53
  _bios_printer 55
  _bios_serialcom 56
  _bios_timeofday 60
  _disable 113
  _dos_freemem 198
  _dos_getdiskfree 122
  _dos_getvect 127
  _dos_keep 340
  _dos_setvect 128
  _enable 113
  _harderr 294
  _hardresume 297
  _hardretn 298
  absread 12
  abswrite 13
  bdos 34
  bdosptr 35
  bioscom 36
  biosdisk 42
  biosequip 45
  bioskey 48
  biosmemory 52
  biosprint 53
  biostime 59
  country 96
  ctrlbrk 107
  disable 113
  dosexterr 118
  enable 113
  FP_OFF 191
  FP_SEG 191
  freemem 198
  geninterrupt 215
  getcbrk 221
  getdfree 231
  getdta 235
  getfat 236
  getfatd 238
  getpsp 264
  getvect 272
  getverify 273
  harderr 290
  hardresume 290
  hardretn 290

inp 314
inportb 315
inpw 316
int86 321
int86x 322
intdos 324
intdosx 325
intr 326
ioctl 327
keep 340
MK_FP 383
outp 400
outport 401
outportb 401
outpw 402
parsfnm 407
peek 408
peekb 410
poke 413
pokeb 414
randbrd 439
segread 479
setcbrk 490
setdta 494
setvect 272
setverify 523
sleep 535
unlink 611
dosexterr (Bibliotheksfunktion) 118
dostounix (Bibliotheksfunktion) 131
drawpoly (Bibliotheksfunktion) 132
dreidimensionale Balken 32
Dreieck
Hypotenuse berechnen 306
druckbare Zeichen
prüfen auf 335
Drucker
direkt drucken 53, 55
installierte 45
Schnittstelle 53, 55
Überprüfung auf Vorhandensein 46
DTA (Disk Transfer Address) 439, 441, 494
ermitteln 235
dup (Bibliotheksfunktion) 134
dup2 (Bibliotheksfunktion) 135
dynamische Speicherverwaltung 549
freie Blöcke des FAR-Heap füllen 158
Überprüfung des FAR-Heap 155, 157
FAR-Heap 151, 153f
Grafikspeicher belegen 286
Grafikspeicher freigeben 284
Speicher freigeben 198
Speicherbelegung auf FAR-Heap 161f
Überprüfung des FAR-Heap 160
Überprüfung des Heap 299ff

E

E2BIG, EACCES...EZERO (Fehler-Konstanten) 636
ecvt (Bibliotheksfunktion) 137
Editieren
Blockoperationen 375f
Ein-/Ausgabe
Bildschirm 97
Diskette 40
formatierte 545, 547
Operationen 42
Puffer 489
Schnittstellen 314ff, 400f
serielle Schnittstelle 36, 56
Steuerung 327
Streams 167, 190, 200, 206, 466, 608
Tastatur 222f, 339, 609
Zeichen auf den Bildschirm 426
siehe auch Ausgabe, Eingabe
Einfügen von Bildschirmzeilen 316
Eingabe
formatierte 105, 466, 547, 618ff
Pufferung 522
siehe auch Ein-/Ausgabe
Eingabefelder von scanf 469
elliptischer Kreisbogen 138
ellipse (Bibliotheksfunktion) 138
Ellipsen
zeichnen und füllen 176
__emit__ (Bibliotheksfunktion) 139
EMS
siehe Expanded Memory
env
Parameter für main 3
environ (globale Variable) 635
Environment
Strings schreiben zu 428
Environment ('Umgebung')
Lesen 235
Übergabe 143, 541
Environment-Variablen 583, 598, 635
eof (Bibliotheksfunktion) 141
errno (globale Variable) 635
erweiterter Fehlercode von DOS 118

Erweiterungsspeicher 406
execl (Bibliotheksfunktion) 142
execle (Bibliotheksfunktion) 142
execlp (Bibliotheksfunktion) 142
execlpe (Bibliotheksfunktion) 142
execv (Bibliotheksfunktion) 142
execve (Bibliotheksfunktion) 142
execvp (Bibliotheksfunktion) 142
execvpe (Bibliotheksfunktion) 142
_exit (Bibliotheksfunktion) 148
exit (Bibliotheksfunktion) 149
Exit-Code 148f
  bei keep und _dos_keep 340
exp (Bibliotheksfunktion) 150
Expanded Memory (EMS) 405f
expl (Bibliotheksfunktion) 150
Exponent
  Berechnung 417f
  double 344
  einer Gleitkommazahl 201
Exponentialfunktion 150

F
FA_ARCH...FA_SYSTEM
    (Konstanten) 180
fabs (Bibliotheksfunktion) 151
fabsl (Bibliotheksfunktion) 151
FAR-Heap 151ff
  freie Blöcke füllen 158
  Knoten überprüfen 157
  Speicher belegen auf 151ff, 161f
  Speicher freigeben 153
  überprüfen und initialisieren 154, 160
  ungenutzer Speicher auf 153
  Zeiger 161f
far-Zeiger 152
  Adreßsegment 191
  erzeugen 383
Farben und Paletten
  ändern 511, 513
  Einträge setzen 481
  Füllfarbe 176f, 182, 412, 477, 497
  Füllmuster 176f, 182, 239f, 412, 477, 496f
  Größe 260
  Hintergrund 219, 485, 590
  höchste Farbnummer 250
  Informationen ermitteln 258
  Pixel 263, 432
  Probleme 19
  Rechteck 451
  Textattribute setzen 588
  Zeichenfarbe setzen 224, 412, 491, 591
  Zeiger auf 230
Farbgrafikadapter (CGA) 412
  Probleme mit 19
  Überprüfung auf Vorhandensein 109
farcalloc (Bibliotheksfunktion) 151
farcoreleft (Bibliotheksfunktion) 153
farfree (Bibliotheksfunktion) 153
farheapcheck (Bibliotheksfunktion) 154
farheapcheckfree
    (Bibliotheksfunktion) 155
farheapchecknode
    (Bibliotheksfunktion) 157
farheapfillfree (Bibliotheksfunktion) 158
farheapwalk (Bibliotheksfunktion) 160
farmalloc (Bibliotheksfunktion) 161
farrealloc (Bibliotheksfunktion) 162
FAT (File Allocation Table)
  Informationen über 236, 238
FCB (File Control Block) 439, 441
fclose (Bibliotheksfunktion) 163
fcloseall (Bibliotheksfunktion) 164
fcvt (Bibliotheksfunktion) 165
fdopen (Bibliotheksfunktion) 166
Fehler
  Behandlung 290, 294, 297
  bei Berechnungen 370
  Bestimmung bei Streams 169
  DOS 118
  erweiterte Bestimmung 118
  Flag 83
  Hardware 290, 294, 297f
  im Modus SHARED 353, 355
  Konstanten und Texte 636
  Lesen/Schreiben 169
  mathematische 370
Fehlermeldungen 559, 561
  ausgeben 411, 635
  des Grafikpakets 283, 288
  _strerror und 561
  strerror und 561
  Zeiger auf 559, 561
Fenster
  Bildlauf ein-/ausschalten 644
feof (Bibliotheksfunktion) 168
ferror (Bibliotheksfunktion) 169
ferror (Bibliotheksmakro) 169
Festplatte
  siehe Disketten

fflush (Bibliotheksfunktion) 169
fgetpos (Bibliotheksfunktion) 172
fgets (Bibliotheksfunktion) 173
filelength (Bibliotheksfunktion) 174
fileno (Bibliotheksfunktion) 175
fillellipse (Bibliotheksfunktion) 176
fillpoly (Bibliotheksfunktion) 177
fillsettingstype (Datentyp) 240
findfirst (Bibliotheksfunktion) 179
findnext (Bibliotheksfunktion) 181
Flächenfüllung 176ff, 182
  Muster ermitteln 239
  Muster setzen 496
  Parameter 240, 497
Flags
  carry 321f, 324f
  Control-Break 107
  Dateiende 83, 168
  Dateizugriff 536
  Fehler 83
  für Bildschirmausgabe 634
  für printf 421
  Lesen/Schreiben 395
  Verify (DOS) 273, 523
floodfill (Bibliotheksfunktion) 182
floor (Bibliotheksfunktion) 184
floorl (Bibliotheksfunktion) 184
flushall (Bibliotheksfunktion) 185
_fmemccpy (Bibliotheksfunktion) 375
_fmemchr (Bibliotheksfunktion) 376
_fmemcmp (Bibliotheksfunktion) 377
_fmemcpy (Bibliotheksfunktion) 378
_fmemicmp (Bibliotheksfunktion) 379
_fmemset (Bibliotheksfunktion) 381
fmod (Bibliotheksfunktion) 186
_fmode (globale Variable) 638
fmodl (Bibliotheksfunktion) 186
fnmerge (Bibliotheksfunktion) 186
fnsplit (Bibliotheksfunktion) 188
Fonts
  Grafik- 268
  festlegen 517
  registrieren 454
  Zeichengröße 596
  siehe auch Vektor-Fonts
fopen (Bibliotheksfunktion) 190
Formatierung
  Ausgabe 97
  cprintf und 97
  fprintf und 194
  fscanf und 202
  grafische Textausgaben 515
  prinf und 418
  scanf und 466
  sprintf und 545
  vfprintf und 616
  vsprinf und 622
  vsscanf und 623
Formatstrings 561
  Ausgabe 419
  in printf 419
  in scanf 467
FP_OFF (Bibliotheksmakro) 192
FP_SEG (Bibliotheksmakro) 192
fpos_t (Datentyp) 173
_fpreset (Bibliotheksfunktion) 192
fprintf (Bibliotheksfunktion) 194
fputchar (Bibliotheksfunktion) 195, 427
fputs (Bibliotheksfunktion) 196
Frame-Zeiger 359
fread (Bibliotheksfunktion) 197
free (Bibliotheksfunktion) 198
freemem (Bibliotheksfunktion) 198
freopen (Bibliotheksfunktion) 200
frexp (Bibliotheksfunktion) 201
frexpl (Bibliotheksfunktion) 201
fscanf (Bibliotheksfunktion) 202
fseek (Bibliotheksfunktion) 203
fsetpos (Bibliotheksfunktion) 205
_fsopen (Bibliotheksfunktion) 206
fstat (Bibliotheksfunktion) 208
_fstrcat (Bibliotheksfunktion) 552
_fstrchr (Bibliotheksfunktion) 553
_fstrcmp (Bibliotheksfunktion) 554
_fstrcpy (Bibliotheksfunktion) 556
_fstrcspn (Bibliotheksfunktion) 557
_fstrdup (Bibliotheksfunktion) 559
_fstricmp (Bibliotheksfunktion) 563
_fstrlen (Bibliotheksfunktion) 564
_fstrlwr (Bibliotheksfunktion) 565
_fstrncat (Bibliotheksfunktion) 565
_fstrncmp (Bibliotheksfunktion) 566
_fstrncpy (Bibliotheksfunktion) 568
_fstrnicmp (Bibliotheksfunktion) 569
_fstrnset (Bibliotheksfunktion) 570
_fstrpbrk (Bibliotheksfunktion) 571
_fstrrchr (Bibliotheksfunktion) 572
_fstrrev (Bibliotheksfunktion) 573
_fstrset (Bibliotheksfunktion) 574
_fstrspn (Bibliotheksfunktion) 574

_fstrstr (Bibliotheksfunktion) 575
_fstrtok (Bibliotheksfunktion) 578
_fstrupr (Bibliotheksfunktion) 581
ftell (Bibliotheksfunktion) 210
ftime (Bibliotheksfunktion) 211
füllen
  Flächen 182
  Polygone 177
_fullpath (Bibliotheksfunktion) 212
Funktionen
  Ergebnis von main 6
fwrite (Bibliotheksfunktion) 213

G

Gameport 45f
gcvt (Bibliotheksfunktion) 214
Genauigkeit
  in Gleitkommaoperationen 92
geninterrupt (Bibliotheksmakro) 215
gerade Parität 37, 56
Gerätetreiber 327
  siehe auch Grafiktreiber
getarccoords (Bibliotheksfunktion) 216
getaspectratio (Bibliotheksfunktion) 218
getbkcolor (Bibliotheksfunktion) 219
getc (Bibliotheksmakro) 221
getcbrk (Bibliotheksfunktion) 221
getchar (Bibliotheksfunktion) 223
getche (Bibliotheksfunktion) 223
getcolor (Bibliotheksfunktion) 224
getcurdir (Bibliotheksfunktion) 226
getcwd (Bibliotheksfunktion) 227
getdate (Bibliotheksfunktion) 228
_getdcwd (Bibliotheksfunktion) 229
getdefaultpalette
    (Bibliotheksfunktion) 230
getdfree (Bibliotheksfunktion) 231
getdisk (Bibliotheksfunktion) 232
_getdrive (Bibliotheksfunktion) 233
getdrivername (Bibliotheksfunktion) 233
getdta (Bibliotheksfunktion) 235
getenv (Bibliotheksfunktion) 235
getfat (Bibliotheksfunktion) 236
getfatd (Bibliotheksfunktion) 238
getfillpattern (Bibliotheksfunktion) 239
getfillsettings (Bibliotheksfunktion) 240
getftime (Bibliotheksfunktion) 243
getgraphmode (Bibliotheksfunktion) 244
getimage (Bibliotheksfunktion) 245
getlinesettings (Bibliotheksfunktion) 248
getmaxcolor (Bibliotheksfunktion) 250
getmaxmode (Bibliotheksfunktion) 252
getmaxx (Bibliotheksfunktion) 253
getmaxy (Bibliotheksfunktion) 254
getmodename (Bibliotheksfunktion) 255
getmoderange (Bibliotheksfunktion) 257
getpalette (Bibliotheksfunktion) 258
getpalettesize (Bibliotheksfunktion) 260
getpass (Bibliotheksfunktion) 261
getpid (Bibliotheksfunktion) 262
getpixel (Bibliotheksfunktion) 263
getpsp (Bibliotheksfunktion) 264
gets (Bibliotheksfunktion) 265
gettext (Bibliotheksfunktion) 266
gettextinfo (Bibliotheksfunktion) 267
gettextsettings (Bibliotheksfunktion) 268
gettime (Bibliotheksfunktion) 271
getvect (Bibliotheksfunktion) 272
getverify (Bibliotheksfunktion) 273
getviewsettings (Bibliotheksfunktion) 274
getx (Bibliotheksfunktion) 277
gety (Bibliotheksfunktion) 278
Gleitkomma-Paket
  _clear87 80
  Re-Initialisierung 192
  Software-Signal 438
  Status-WORD 92, 550
Gleitkommazahlen
  absoluten Wert berechnen 151
  BCD 33, 450
  in Strings konvertieren 165
  Strings und 27
globale Variablen 633ff
gmtime (Bibliotheksfunktion) 279
goto, nicht-lokales 358
gotoxy (Bibliotheksfunktion) 280
Grafikadapter 109
  Monochrom 19
Grafikkarten
  untersuchen 310
Grafikmodus 109, 244, 252
  erlaubter Bereich 257
  ermitteln 244
  Name 255
  setzen 501
Grafikpaket
  beenden 88
  Fehlerbehandlung 283, 288
  Heap-Verwaltung 284, 286
  Initialisierung 310

Puffergröße 500
Registrierung von Treibern 452
Registrierung von Zeichensätzen 454
Standardvorgaben 281
Grafikpuffer
Größe 500
Grafikroutinen
arc 19
bar 30
bar3d 32
circle 79
cleardevice 81
clearviewport 84
closegraph 88
detectgraph 109
drawpoly 132
ellipse 138
fillellipse 176
fillpoly 177
floodfill 182
getarccoords 216
getaspectratio 218
getbkcolor 219
getcolor 224
getdefaultpalette 230
getdrivername 233
getfillpattern 239
getfillsettings 240
getgraphmode 244
getimage 245
getlinesettings 248
getmaxcolor 250
getmaxmode 252
getmaxx 253
getmaxy 254
getmodename 255
getmoderange 257
getpalette 258
getpalettesize 260
getpixel 263
gettextsettings 268
getviewsettings 274
getx 277
gety 278
graphdefaults 281
grapherrormsg 283
_graphfreemem 284
_graphgetmem 286
graphresult 288
imagesize 308
initgraph 310
installuserfont 320
line 347
linerel 348
lineto 349
moverel 388
moveto 390
outtext 402
outtextxy 404
pieslice 412
putimage 429
putpixel 432
registerbgidriver 452
registerbgifont 454
restorecrtmode 458
sector 477
setactivepage 479
setallpalette 481
setaspectratio 484
setbkcolor 485
setcolor 491
setfillpattern 496
setfillstyle 497
setgraphbufsize 500
setgraphmode 501
setlinestyle 505
setpalette 511
setrgbpalette 513
settextjustify 515
settextstyle 517
setusercharsize 520
setviewport 524
setvisualpage 526
setwritemode 527
textheight 593
textwidth 596
Grafikspeicherseite 526
Grafiktreiber 109
BGI und 288, 310, 317
Konstanten in graphics.h 110, 310
laden 310
Modus 252, 255, 257, 266, 310, 458
Modus ermitteln 252
Namen ermitteln 233
neuer 317
registrieren 452
graphdefaults (Bibliotheksfunktion) 281
grapherrormsg (Bibliotheksfunktion) 283
_graphfreemem (Bibliotheksfunktion) 284
_graphgetmem (Bibliotheksfunktion) 286

graphresult (Bibliotheksfunktion) 288
Greenwich Mean Time (GMT) 106, 211, 352
  konvertieren zu 279
  setzen 604
  timezone (glob. Variable) 643
Großbuchstaben
  konvertieren zu 602f
  prüfen auf 337
  zu Kleinbuchstaben 565, 601
Größe
  Datei 78, 174
  Datensegment 465
  Farbpalette 260
  NEAR-Heap 639
  Speicher 46, 48, 52f
  von Stack und Heap 642
  von Textausgaben 517

H
Handles
  Datei- 116, 134f, 166, 175
_harderr (Bibliotheksfunktion) 294
harderr (Bibliotheksfunktion) 290
_hardresume (Bibliotheksfunktion) 297
hardresume (Bibliotheksfunktion) 290
_hardretn (Bibliotheksfunktion) 298
hardretn (Bibliotheksfunktion) 290
Hardware
  Druckerschnittstelle 53, 55
  Ein-/Ausgabe 327
  Fehlerbehandlung 290, 294, 297f
  Gerätetyp prüfen 331
  Interrupts aktivieren/deaktivieren 113
  Schnittstellen 314ff, 400ff
  Überprüfung auf Interrupts 45
Heap 93
  FAR 151ff
  freie Blöcke füllen 303
  Größe festlegen 639, 642
  Knoten überprüfen 301
  NEAR 639
  Speicher reservieren 368
  überprüfen 299ff, 304
  Verwaltung 66
  Verwaltung durch das Grafikpaket 286
  siehe auch FAR-Heap, NEAR-Heap
heapcheck (Bibliotheksfunktion) 299
heapcheckfree (Bibliotheksfunktion) 300
heapchecknode (Bibliotheksfunktion) 301
_HEAPEMPTY 304
_HEAPEND 304
heapfillfree (Bibliotheksfunktion) 303
_heaplen (globale Variable) 639
_HEAPOK 304
heapwalk (Bibliotheksfunktion) 304
hexadezimale Zahlen
  prüfen auf 338
highvideo (Bibliotheksfunktion) 305
Hintergrundfarbe
  ermitteln 219
  für Textausgaben 590
  setzen 485
  siehe auch Farben und Paletten
hohe Intensität (Zeichen) 305
Höhe von Textausgaben 593
Höhen-/Seitenverhältnis (Bildschirm) 218, 593
  ändern 484
hyperbolische Funktionen 95, 534, 585
hyperbolischer Cosinus 95
hypot (Bibliotheksfunktion) 306
Hypotenuse
  berechnen 306
hypotl (Bibliotheksfunktion) 306

I
I/O
  siehe Ein-/Ausgabe
I/O-Routinen
  _chmod 75
  _close 86
  _creat 99
  _dos_creat 99
  _dos_creatnew 117
  _dos_open 394
  _dos_read 444
  _dos_write 130
  _fsopen 206
  _open 394
  _read 444
  _write 628
  access 14
  cgets 70
  chmod 77
  clearerr 83
  close 86
  clreol 89
  clrscr 90
  cprintf 97

creat 101
creatnew 102
creattemp 104
cscanf 105
delline 109
dup 134
dup2 135
eof 141
fclose 163
fcloseall 164
feof 168
ferror 169
fflush 169
fgetc 171
fgetchar 172
fgets 173
filelength 174
fileno 175
flushall 185
fopen 190
fprintf 194
fputc 195
fputchar 195, 427
fputs 196
fread 197
freopen 200
fscanf 202
fseek 203
fstat 208
ftell 210
fwrite 213
getc 221
getch 222
getche 223
getpass 261
gets 265
ioctl 327
isatty 331
kbhit 339
lock 353
locking 355
lseek 365
open 396
perror 411
printf 418
putc 426
putch 426
puts 433
putw 435
read 447
remove 456
rename 457
rewind 459
scanf 466
setbuf 489
setftime 243
setmode 509
setvbuf 521
sopen 536
sprintf 545
sscanf 547
stat 208
tell 586
umask 606
unlock 612
vfprintf 616
vfscanf 618
vprintf 619
vscanf 620
vsprintf 622
vsscanf 623
write 629
imag (Bibliotheksfunktion) 307
imagesize (Bibliotheksfunktion) 308
Imaginärteil 91, 307
initgraph (Bibliotheksfunktion) 310
Initialisierung
  Grafikpaket 310
  Speicher 381, 508
  Zufallszahlen-Generator 444, 547
inp (Bibliotheksmakro) 314
inport (Bibliotheksfunktion) 314
inportb (Bibliotheksmakro) 315
inpw (Bibliotheksmakro) 316
insline (Bibliotheksfunktion) 316
Installation
  von Vektor-Fonts 320
installuserdriver (Bibliotheksfunktion) 317
installuserfont (Bibliotheksfunktion) 320
int (Datentyp)
  Konvertierung nach String 606
  Vertauschung der Worthälften 583
int86 (Bibliotheksfunktion) 321
int86x (Bibliotheksfunktion) 322
intdos (Bibliotheksfunktion) 324
intdosx (Bibliotheksfunktion) 325
Integer
  Absolutwert 11
  Ausgabe zu Streams 435
  bitweise rotieren 463f

dividieren 345
konvertieren zu Strings 339
lesen aus Streams 275
long 344
speichern 413
Intensität (Zeichen)
hohe 305
niedrige 360
normal 393
Interpunktionszeichen
prüfen auf 336
Interrupt-Handler 71
DOS 107
Signal-Handler und 531
Interrupt-Vektoren 107
setzen und lesen 127f, 272
Interrupts
Control-Break 107
DOS 290, 294
erzeugen 215
Hardware 113
NMI (non-maskable) 114
Software 215, 321ff
Software-Signal 438
Strg-Untbr 221
siehe auch DOS-Interrupts
intr (Bibliotheksfunktion) 326
ioctl (Bibliotheksfunktion) 327
IOFBF_, IOLBF_, IONBF_ (Konstanten) 522
isalnum (Bibliotheksmakro) 329
isalpha (Bibliotheksmakro) 330
isascii (Bibliotheksmakro) 331
isatty (Bibliotheksfunktion) 331
iscntrl (Bibliotheksmakro) 332
isdigit (Bibliotheksmakro) 333
isgraph (Bibliotheksmakro) 334
islower (Bibliotheksmakro) 334
isprint (Bibliotheksmakro) 335
ispunct (Bibliotheksmakro) 336
isspace (Bibliotheksmakro) 336
isupper (Bibliotheksmakro) 337
isxdigit (Bibliotheksmakro) 338
itoa (Bibliotheksfunktion) 339

J

Japanisches Datumsformat 97
Jokerzeichen
in Kommandozeilen 5
Justierung
von Textausgaben 515

K

Kalenderformat 385
kbhit (Bibliotheksfunktion) 339
keep (Bibliotheksfunktion) 340
Klassifizierungsroutinen
isalnum 329
isalpha 330
isascii 331
iscntrl 332
isdigit 333
isgraph 334
islower 334
isprint 335
ispunct 336
isspace 336
isupper 337
isxdigit 338
Kleinbuchstaben
zu Großbuchstaben 581, 601ff
prüfen auf 334
Kommandozeilencompiler
-p (Pascal-Aufrufkonvention) 6
Kommandozeilenparameter 633
Beispielprogramm 4
Jokerzeichen und 5
komplexe Zahlen 392
Absolutwert berechnen 64
Imaginärteil 307
konjugierte 92
Konstruktor für 91
polar-Funktion 415
Realteil 449
Winkel 20
konjugierte komplexe Zahl 92
Konstanten
CLK_TCK 85
FA_ARCH...FA_SYSTEM 180
IOFBF_, IOLBF_ und IONBF 522
P_WAIT, P_NOWAIT, P_OVERLAY 540
S_IWRITE, S_IREAD 77
SEEK_CUR, SEEK_END und SEEK_SET 204
konvertieren
Datum und Uhrzeit 21, 106, 131, 279, 352, 385
DOS- in UNIX-Format 131
double in Ganzzahl und Bruchteil 386
Format-Spezifizierer 419

Gleitkommazahl in String 137, 165, 214
Groß- in Kleinbuchstaben 565, 601
Integer in String 339
Klein- in Großbuchstaben 581, 602f
long-Wert in String 366
String in Integer 28
String in long-Wert 29
String in double 577
String in long 579
String in long double 577
String in unsigned long 580
String in Gleitkommazahl 27
Uhrzeit in String 576
UNIX- in DOS-Format 610
unsigned long in String 606
Zahl in BCD-Zahl 33
Zahl in String 339
Zeichen in ASCII-Format 600
Konvertierungsanweisungen
(für printf) 419
Konvertierungsroutinen
_atold 27
atof 27
atoi 28
atol 29
ecvt 137
fcvt 165
gcvt 214
itoa 339
ltoa 366
strtod 577
strtol 580
toascii 600
_tolower 601
tolower 601
_toupper 602
toupper 603
ultoa 606
Koordinaten
aktuelle Position (CP) 277f, 524
Cursorposition 280, 626
maximale 253f
von arc ermitteln 216
kopieren
Daten 387
Koprozessor
siehe mathematischer Koprozessor
Korrekturfaktor für Höhen-
/Seitenverhältnis 484
Kreisbögen 19, 138, 477
Kreise 79, 176
Kuchenstück 412, 477

L

labs (Bibliotheksfunktion) 344
landesspezifische Formate 96
Länge
von Dateien 78, 174
von Strings 564
Laufwerk
aktuelle Nummer 123
aktuelles Verzeichnis 226
Funktionen 42
Nummer 232f
setzen 74
Überprüfung auf Vorhandensein 45
Lautsprecher
aktivieren 539
ausschalten 393
ldexp (Bibliotheksfunktion) 344
ldexpl (Bibliotheksfunktion) 344
ldiv (Bibliotheksfunktion) 345
Leerzeichen
prüfen auf 336
Lesefehler 169
lesen
von der Tastatur 70
Lese-/Schreibzugriff 606
prüfen auf 14
lfind (Bibliotheksfunktion) 346
line (Bibliotheksfunktion) 347
lineare Suche 346, 363
linerel (Bibliotheksfunktion) 348
linesettingstype (Datentyp) 248, 505
lineto (Bibliotheksfunktion) 349
Linien
Art 248, 505
Dicke 248, 505
Muster 248
Parameter 527
Rechtecke und 451
Schreibmodus 527
zeichnen 347ff
local-Variable
auswählen 507
localeconv (Bibliotheksfunktion) 351
localtime (Bibliotheksfunktion) 352
lock (Bibliotheksfunktion) 353
locking (Bibliotheksfunktion) 355
log (Bibliotheksfunktion) 356

log10 (Bibliotheksfunktion) 357
log10l (Bibliotheksfunktion) 357
Logarithmus
  natürlicher 356
  zur Basis 10 357
logl (Bibliotheksfunktion) 356
lokale Struktur
  Zeiger auf 351
long-Wert
  zu String konvertieren 366
longjmp (Bibliotheksfunktion) 358
löschen
  Bildschirm 501
  Bildschirm im Textmodus 90
  Bildschirm in der Grafik 81
  bis zum Zeilenende 89
  Dateien 456, 611
  Textzeilen 109
  Verzeichnis 461
  Zeichenfenster 84
  Zeile in Textfenster 89
lowvideo (Bibliotheksfunktion) 360
_lrotl (Bibliotheksfunktion) 361
_lrotr (Bibliotheksfunktion) 362
lsearch (Bibliotheksfunktion) 363
lseek (Bibliotheksfunktion) 365
ltoa (Bibliotheksfunktion) 366

M

main-Funktion 3
  als C-Routine deklarieren 6
  Argumente 633
  Beispiel 4
  Ergebnis 6
  Jokerzeichen 5
  Parameter 3
_makepath (Bibliotheksfunktion) 367
Makros
  abs 12
  assert 23f
  Ausgabe zu Schnittstellen 400ff
  far-Zeiger erstellen 383
  ferror 169
  fileno 175
  FP_OFF 192
  FP_SEG 192
  geninterrupt 215
  getc 221
  getchar 223
  isalnum 329
  isalpha 330
  isascii 331
  iscntrl 332
  isdigit 333
  isgraph 334
  islower 334
  isprint 335
  ispunct 336
  isspace 336
  isupper 337
  isxdigit 338
  max 372
  min 381
  MK_FP 383
  outp 400
  peek 408
  peek 408
  peekb 410
  peekb 410
  poke 413
  poke 414
  pokeb 414
  pokeb 414
  putc 426
  putc 426
  putchar 427
  random 443
  randomize 444
  remove 456
  sopen 536
  strcmpi 555
  strncmpi 567
  toascii 600
  _tolower 601
  _toupper 602
  va_arg 615
  va_end 615
  va_start 615
  Vergleich zweier Werte 372, 381
malloc (Bibliotheksfunktion) 368
Mantisse 386
  einer Gleitkommazahl 201
Maschinensprache
  Literale in Objektcode einfügen 139
mathematische Routinen
  abs 11
  acos 15
  asin 22
  asinl 22
  atan 24

atan2 25
atan2l 25
atanl 24
atof 27
atoi 28
atol 29
_atold 27
bcd 33
cabs 64
cabsl 64
ceil 67
ceill 67
clear87_ 80
control87_ 92
cos 94
cosh 95
coshl 95
cosl 94
div 115
ecvt 137
exp 150
expl 150
fabs 151
fabsl 151
fcvt 165
floor 184
floorl 184
fmod 186
fmodl 186
_fpreset 192
frexp 201
frexpl 201
gcvt 214
hypot 306
hypotl 306
itoa 339
labs 344
ldexp 344
ldexpl 344
ldiv 345
log 356
log10 357
log10l 357
logl 356
_lrotl 361
_lrotr 362
ltoa 366
matherr 370
_matherrl 370
modf 386
modfl 386
poly 416
polyl 416
pow 417
pow10 418
pow10l 418
powl 417
rand 439
_rotl 463
_rotr 464
sin 534
sinh 534
sinhl 534
sinl 534
sqrt 546
sqrtl 546
srand 547
_status87 550
strtod 577
strtol 580
tan 584
tanh 585
tanhl 585
tanl 584
ultoa 606
mathematischer Koprozessor 45, 192, 550, 633
  Ausnahmezustände 81
  Status-WORD 80, 92
  Überprüfung auf Vorhandensein 46, 48
matherr (Bibliotheksfunktion) 370
_matherrl (Bibliotheksfunktion) 370
max (Bibliotheksmakro) 372
mblen (Bibliotheksfunktion) 373
mbstowcs (Bibliotheksfunktion) 374
mbtowc (Bibliotheksfunktion) 374
memccpy (Bibliotheksfunktion) 375
memchr (Bibliotheksfunktion) 376
memcmp (Bibliotheksfunktion) 377
memcpy (Bibliotheksfunktion) 378
memicmp (Bibliotheksfunktion) 379
memmove (Bibliotheksfunktion) 380
memset (Bibliotheksfunktion) 381
MEZ 106
min (Bibliotheksmakro) 381
MK_FP (Bibliotheksmakro) 383
mkdir (Bibliotheksfunktion) 382
mktemp (Bibliotheksfunktion) 384
mktime (Bibliotheksfunktion) 385
modf (Bibliotheksfunktion) 386

modfl (Bibliotheksfunktion) 386
Modulo 186
Modus
  Datei 101f, 117, 124, 167, 190, 200
Monochrom-Grafikadapter 19, 109, 412
movedata (Bibliotheksfunktion) 387
moverel (Bibliotheksfunktion) 388
movetext (Bibliotheksfunktion) 389
moveto (Bibliotheksfunktion) 390
movmem (Bibliotheksfunktion) 391

N
NaN (Not a Number)
  atof und 28
nationale Formate 96
natürlicher Logarithmus 356
NDEBUG-Symbol 24
NEAR-Heap
  Größe 639
neue Dateien 99, 101f, 117
_new_handler (globale Variable) 640
nicht-lokales goto 358
niedrige Intensität (Zeichen) 360
norm (Bibliotheksfunktion) 392
normale Intensität (Zeichen) 393
normvideo (Bibliotheksfunktion) 393
nosound (Bibliotheksfunktion) 393

O
Objektcode
  Literale im 139
Öffnen
  von Dateien 190, 394, 399, 536
Offset
  far-Zeiger 383
  von far-Adressen 191
_open (Bibliotheksfunktion) 394
open (Bibliotheksfunktion) 396
opendir (Bibliotheksfunktion) 399
_osmajor (globale Variable) 640
_osminor (globale Variable) 640
outp (Bibliotheksmakro) 400
outport (Bibliotheksfunktion) 401
outportb (Bibliotheksfunktion) 401
outpw (Bibliotheksfunktion) 402
outtext (Bibliotheksfunktion) 402
outtextxy (Bibliotheksfunktion) 404
Overlay-Puffer
  Größe ändern 641
Overlays 406
  Coroutinen und 358
  Expanded Memory und 405
_ovrbuffer (globale Variable) 641
_OvrInitEms (Bibliotheksfunktion) 405
_OvrInitExt (Bibliotheksfunktion) 406

P
-p (Pascal-Aufrufkonvention) 6
P_WAIT, P_NOWAIT, P_OVERLAY
    (Konstanten) 540
Paletten
  siehe Farben und Paletten
palettetype (Datentyp) 259, 481
Parameter
  für main 3
  Kommandozeile 633
  variable Anzahl 615
parent-Prozeß 142, 540
Parität 37, 56
Paritätsprüfung 37, 56
parsfnm (Bibliotheksfunktion) 407
Pascal-Aufrufkonventionen (-p) 6
Paßwörter 261
PATH
  (DOS-Environment-Variable) 142, 541
Pause
  Programmausführung 535
peek (Bibliotheksfunktion) 408
peek (Bibliotheksmakro) 408
peekb (Bibliotheksfunktion) 410
peekb (Bibliotheksmakro) 410
Peripheriegeräte
  Liste ausgeben 45
  siehe Hardware 327
perror (Bibliotheksfunktion) 411, 635
Pfadnamen
  erzeugen 186, 367
  relative konvertieren 212
  suchen 227
  zerlegen 188, 543
pieslice (Bibliotheksfunktion) 412
Pixel
  Farbe 263, 432
poke (Bibliotheksfunktion) 413
poke (Bibliotheksmakro) 414
pokeb (Bibliotheksfunktion) 414
pokeb (Bibliotheksmakro) 414
polar (Bibliotheksfunktion) 415
poly (Bibliotheksfunktion) 416
Polygone

zeichnen 132
zeichnen und füllen 177
polyl (Bibliotheksfunktion) 416
Polynome 416
Ports
siehe Schnittstellen
Positionierung in einer Datei 459
pow (Bibliotheksfunktion) 417
pow10 (Bibliotheksfunktion) 418
pow10l (Bibliotheksfunktion) 418
powl (Bibliotheksfunktion) 417
printf (Bibliotheksfunktion) 418
Programme
Aufräumarbeiten 68
Ausführung verzögern 108
beenden 148f, 340
Jokerzeichen beim Aufruf 5
laden und starten 142
pausieren 535
Prozeß-ID 262
speicherresidente 71, 340
starten 142
Programmsegmentpräfix (PSP) 262, 264
des aktuellen Programms 642
Größe 639
Speichermodelle und 642
Prozeß-ID 262
Prozeßroutinen
getpid 262
_c_exit 68
_cexit 69
_exit 148
abort 11
execl 142
execle 142
execlp 142
execlpe 142
execv 142
execve 142
execvp 142
execvpe 142
exit 149
spawnl 539
spawnle 539
spawnlp 539
spawnlpe 539
spawnv 539
spawnve 539
spawnvp 539
spawnvpe 539
system 583
Prozesse
übergeordnete 142, 540
untergeordnete 142, 540
_psp (globale Variable) 264, 642
Puffer
leeren 185
Overlay 641
Streams und 163f, 489, 521
Tastatur 609
Puffergröße des Grafikpakets 500
Pufferung von Ein-/Ausgaben 522
putc (Bibliotheksmakro) 426
putch (Bibliotheksfunktion) 426
putenv (Bibliotheksfunktion) 428
putimage (Bibliotheksfunktion) 429
putpixel (Bibliotheksfunktion) 432
putputchar (Bibliotheksmakro) 427
puts (Bibliotheksfunktion) 433
puttext (Bibliotheksfunktion) 434
putw (Bibliotheksfunktion) 435

Q

qsort (Bibliotheksfunktion) 436
Quadratwurzel 150, 357f, 546
QuickSort-Algorithmus 436
Quotient 115, 345

R

rad (Winkeleinheit) 16, 25
raise (Bibliotheksfunktion) 438
RAM
Größe 46, 48, 52f
residente Programme 340
ungenutzer 93
rand (Bibliotheksfunktion) 439
randbrd (Bibliotheksfunktion) 439
randbwr (Bibliotheksfunktion) 441
random (Bibliotheksmakro) 443
Random-Record-Feld 439, 441
randomize (Bibliotheksmakro) 444
_read (Bibliotheksfunktion) 444
read (Bibliotheksfunktion) 447
readdir (Bibliotheksfunktion) 448
real (Bibliotheksfunktion) 449
realloc (Bibliotheksfunktion) 450
Realteil 91, 449
Rechtecke 451
Records
Random 439, 441

rectangle (Bibliotheksfunktion) 451
Rechteck zeichnen 451
Register
  Segmentregister lesen 479
registerbgidriver
    (Bibliotheksfunktion) 452
registerbgifont (Bibliotheksfunktion) 454
Registervariablen 359
  Task-Status und 503
Registrierung
  Grafiktreiber und Zeichensätze 452, 454
REGPACK-Struktur 326
remove (Bibliotheksfunktion) 456
rename (Bibliotheksfunktion) 457
Rest
  siehe Divisionsrest
restorecrtmode (Bibliotheksfunktion) 458
rewind (Bibliotheksfunktion) 459
rewinddir (Bibliotheksfunktion) 460
rmdir (Bibliotheksfunktion) 461
rmtmp (Bibliotheksfunktion) 462
Rotation
  long int 361
Rotation, bitweise 361f, 463f
_rotl (Bibliotheksfunktion) 463
_rotr (Bibliotheksfunktion) 464
RS-232 36, 56
Rundung 34, 67, 184
Rundungsrichung
  in Gleitkommaoperationen 92

S

S_IFCHR, S_IFREG (Konstanten) 208
S_IREAD 606
S_IREAD, S_IWRITE (Konstanten) 77,
    208, 606
sbrk (Bibliotheksfunktion) 465
scanf (Bibliotheksfunktion) 466
schließen
  Datei 86, 116
  Verzeichnis-Stream 87
Schnittstellen
  Ein-/Ausgabe 36, 56, 314ff, 400ff
  serielle 36, 56
  Überprüfung auf Vorhandensein 45f
Schreibrichtung
  festlegen 517
_searchenv (Bibliotheksfunktion) 475
searchpath (Bibliotheksfunktion) 476
sector (Bibliotheksfunktion) 477
SEEK_CUR, SEEK_END und SEEK_SET
    (Konstanten) 204
Segmentpräfix
  siehe Programmsegmentpräfix
Segmente
  far-Zeiger 191, 383
  Register lesen 479
segread (Bibliotheksfunktion) 479
serielle Schnittstelle
  Ein-/Ausgabe 36, 56
set_new_handler
    (Bibliotheksfunktion) 509
setactivepage (Bibliotheksfunktion) 479
setallpalette (Bibliotheksfunktion) 481
setaspectratio (Bibliotheksfunktion) 484
setbkcolor (Bibliotheksfunktion) 485
setblock (Bibliotheksfunktion) 487
setbuf (Bibliotheksfunktion) 489
setcbrk (Bibliotheksfunktion) 490
setcolor (Bibliotheksfunktion) 491
  größter Farbwert für 250
_setcursortype (Bibliotheksfunktion) 493
setdate (Bibliotheksfunktion) 228
setdisk (Bibliotheksfunktion) 232
setdta (Bibliotheksfunktion) 494
setfillpattern (Bibliotheksfunktion) 496
setfillstyle (Bibliotheksfunktion) 497
setftime (Bibliotheksfunktion) 243
setgraphbufsize (Bibliotheksfunktion) 500
setgraphmode (Bibliotheksfunktion) 501
setjmp (Bibliotheksfunktion) 503
setlinestyle (Bibliotheksfunktion) 505
setlocale (Bibliotheksfunktion) 507
setmem (Bibliotheksfunktion) 508
setmode (Bibliotheksfunktion) 509
_set_new_handler
    (Bibliotheksfunktion) 640
setpalette (Bibliotheksfunktion) 511
setrgbpalette (Bibliotheksfunktion) 513
settextjustify (Bibliotheksfunktion) 515
settextstyle (Bibliotheksfunktion) 517
settime (Bibliotheksfunktion) 271
setusercharsize (Bibliotheksfunktion) 520
setvbuf (Bibliotheksfunktion) 521
setvect (Bibliotheksfunktion) 272
setverify (Bibliotheksfunktion) 523
setviewport (Bibliotheksfunktion) 524
setvisualpage (Bibliotheksfunktion) 526
setwritemode (Bibliotheksfunktion) 527
SHARED-Modus 206, 353, 395, 536

sichtbare Grafikseite 526
sig_atomic_t (Datentyp) 530
signal (Bibliotheksfunktion) 529
Signal-Handler 438, 529
sin (Bibliotheksfunktion) 534
sinh (Bibliotheksfunktion) 534
sinhl (Bibliotheksfunktion) 534
sinl (Bibliotheksfunktion) 534
Sinus 534
  hyperbolicus 534
sleep (Bibliotheksfunktion) 535
Software-Interrupts 321, 323, 326
  erzeugen 215
Software-Signale
  senden 438
  signal 529
Sommerzeit 634
sopen (Bibliotheksmakro) 536
sortieren
  Array-Elemente 436
  QuickSort 436
sound (Bibliotheksfunktion) 539
spawnl (Bibliotheksfunktion) 539
spawnle (Bibliotheksfunktion) 539
spawnlp (Bibliotheksfunktion) 539
spawnlpe (Bibliotheksfunktion) 539
spawnv (Bibliotheksfunktion) 539
spawnve (Bibliotheksfunktion) 539
spawnvp (Bibliotheksfunktion) 539
spawnvpe (Bibliotheksfunktion) 539
Speicher
  belegen 66
  belegen auf FAR-Heap 151, 161f
  Bildausschnitt speichern 245, 308
  Block kopieren 375, 378, 380
  Daten kopieren 387
  Daten verlagern 391
  DMA 44ff, 48
  DTA 235
  FAR-Heap 158
  FAR-Heap überprüfen 154f, 157, 160
  freie Blöcke füllen 303
  freie Blöcke überprüfen 300
  freigeben 198, 284
  freigeben auf FAR-Heap 153
  Größe 52f, 549
  initialisieren 381, 508
  Speichermodelle und 66
  ungenutzer 93
  ungenutzer auf FAR-Heap 153
  Zugriff auf 44
  siehe auch dynamische Speicherverwaltung
Speicherbelegung
  Datensegment 61
  Fehler 640, 509
  siehe auch dynamische Speicherverwaltung
Speicherblöcke
  freie füllen 158
  Größe ändern 162, 450, 487
  initialisieren 508
  random 439, 441
  suchen 377
Speichermodelle
  DOS-Systemaufrufe und 34f
  Größe des NEAR-Heap 639
  Heap und 66
  Programmsegmentpräfix und 642
  Speicherreservierung und 161f
  Stack und 642
Speicherreservierung, dynamische 549
speicherresidente Programme 71, 340
Sperre
  entfernen 612
_splitpath (Bibliotheksfunktion) 543
sprintf (Bibliotheksfunktion) 545
Sprünge
  longjmp 358
  setjmp 503
sqrt (Bibliotheksfunktion) 546
sqrtl (Bibliotheksfunktion) 546
srand (Bibliotheksfunktion) 547
sscanf (Bibliotheksfunktion) 547
Stack 66, 93
  Größe festlegen 549, 639, 642
  Speicher belegen auf 16
  Zeiger 359, 503
stackavail (Bibliotheksfunktion) 549
Standardfarbpalette 230
Standardroutinen
  abort 11
  abs 11
  atexit 26
  atof 27
  atoi 28
  atol 29
  _atold 27
  bsearch 62
  calloc 66

_c_exit 68
_cexit 69
ecvt 137
_exit 148
exit 149
fcvt 165
free 198
gcvt 214
getenv 235
itoa 339
labs 344
lfind 346
lsearch 363
ltoa 366
malloc 368
putenv 428
qsort 436
rand 439
realloc 450
srand 547
strtod 577
strtol 580
swab 583
system 583
ultoa 606
stat (Bibliotheksfunktion) 208
stat-Struktur 208
_status87 (Bibliotheksfunktion) 550
Status-WORD 92, 550
löschen 80
Statusbits 37, 56
stdin
formatierte Eingabe von 620
lesen und formatieren aus 466
Puffer und 489
String lesen aus 265
Zeichen lesen aus 172, 223
stdout
formatierte Ausgabe zu 419, 619
Puffer und 489
Strings ausgeben zu 433
Zeichen ausgeben zu 195, 427
stime (Bibliotheksfunktion) 550
_stklen (globale Variable) 642
Stopbits 37, 56
stpcpy (Bibliotheksfunktion) 551
strcat (Bibliotheksfunktion) 552
strchr (Bibliotheksfunktion) 553
strcmp (Bibliotheksfunktion) 554
strcmpi (Bibliotheksfunktion) 555
strcmpi (Bibliotheksmakro) 555
strcoll (Bibliotheksfunktion) 556
strcpy (Bibliotheksfunktion) 556
strcspn (Bibliotheksfunktion) 557
_strdate (Bibliotheksfunktion) 558
strdup (Bibliotheksfunktion) 559
Streams
Ausgabepuffer in Datei schreiben 169
Datei-Handle zuordnen 166
Dateiende-Flag zurücksetzen 83
Dateizeiger 203, 205
Ein-/Ausgabe 167, 190, 200, 206
Ende überprüfen 168
Fehler-Flag zurücksetzen 83
formatierte Ausgabe zu 194, 419, 616, 619
formatierte Eingabe aus 202, 618, 620
Handle 175
Integer lesen aus 275
Integer ausgeben zu 435
lesen aus 171ff, 197, 202, 265
lesen und formatieren aus 466
öffnen 190, 200, 206
Puffer leeren 185
Puffer zuordnen 521
schließen 163f, 200
schreiben in 213
Strings ausgeben zu 196, 433
ungepufferte 489
Zeichen ausgeben zu 195, 426
Zeichen lesen aus 221, 608
Zeiger initialisieren 459
_strerror (Bibliotheksfunktion) 559
strerror (Bibliotheksfunktion) 561
strftime (Bibliotheksfunktion) 561
Strg-Untbr 490
stricmp (Bibliotheksfunktion) 563
Strings
absuchen 571f, 578
anzeigen 98
aus Environment-Tabelle lesen 235
ausgeben zu stdout 433
ausgeben zu einen Stream 196
Breite ermitteln 596
formatierte Ausgabe in 545, 622
formatierte Eingabe aus 623
Gleitkommazahl konvertieren zu 214
Klein- zu Großbuchstaben 581
kopieren 551, 556, 559, 568
konvertieren zu Datum 558

konvertieren zu double 577
konvertieren zu Gleitkommazahlen 165
konvertieren zu long 579
konvertieren zu long double 577
konvertieren zu unsigned long 580
Länge 564
Länge eines Teilstrings 557
lesen aus stdin 265
lesen aus Streams 173
lesen und formatieren aus 547
lesen von Tastatur 70
mit Fehlermeldungen 559, 561
mit Zeichen füllen 570
Teilstring kopieren 582
Teilstring suchen 574f
überschreiben 574
Uhrzeit konvertieren zu 576
umdrehen 573
unsigned long zu 606
vergleichen 377, 379, 554ff, 563, 566f, 569
verketten 552, 565
Zeichen suchen in 553
strlen (Bibliotheksfunktion) 564
strlwr (Bibliotheksfunktion) 565
strncat (Bibliotheksfunktion) 565
strncmp (Bibliotheksfunktion) 566
strncmpi (Bibliotheksmakro) 567
strncpy (Bibliotheksfunktion) 568
strnicmp (Bibliotheksfunktion) 569
strnset (Bibliotheksfunktion) 570
strpbrk (Bibliotheksfunktion) 571
strrchr (Bibliotheksfunktion) 572
strrev (Bibliotheksfunktion) 573
strset (Bibliotheksfunktion) 574
strspn (Bibliotheksfunktion) 574
strstr (Bibliotheksfunktion) 575
_strtime (Bibliotheksfunktion) 576
strtod (Bibliotheksfunktion) 577
strtok (Bibliotheksfunktion) 578
strtol (Bibliotheksfunktion) 579f
_strtold (Bibliotheksfunktion) 577
strtoul (Bibliotheksfunktion) 580
struct _complex1 (Datentyp) 64
struct complex (Datentyp) 64
struct COUNTRY (Datentyp) 96
struct date (Datentyp) 228
struct dfree (Datentyp) 231
struct diskfree_t (Datentyp) 122
struct dosdate_t (Datentyp) 228
struct DOSERROR (Datentyp) 118
struct dostime_t (Datentyp) 126
struct exception (Datentyp) 370
struct fatinfo (Datentyp) 237f
struct ffblk (Datentyp) 179
struct find_t (Datentyp) 119
struct ftime (Datentyp) 243
struct REGPACK (Datentyp) 326
struct stat (Datentyp) 208
struct time (Datentyp) 271
struct timeb (Datentyp) 211
struct tm (Datentyp) 279, 352
struct utimbuf (Datentyp) 614
Struktur
lokale 351
strupr (Bibliotheksfunktion) 581
strxfrm (Bibliotheksfunktion) 582
Suche
in Speicherblöcken 377
in Strings 571f
lineare 346, 363
Suchwege
siehe Pfadnamen
swab (Bibliotheksfunktion) 583
Syntaxfehler
siehe Fehler 583
sys_errlist (globale Variable) 635
sys_nerr (globale Variable) 635
System
Datum und Uhrzeit 126, 271, 550, 597
Fehlermeldungen 411, 635
Puffer 163f
system (Bibliotheksfunktion) 583

T

Tabellen
absuchen 62, 363
tan (Bibliotheksfunktion) 584
Tangens 584
Tangens hyperbolicus 585
tanh (Bibliotheksfunktion) 585
tanhl (Bibliotheksfunktion) 585
tanl (Bibliotheksfunktion) 584
Task-Status 358, 503
Tastatur
Abfrage 48, 50
Ein-/Ausgabe 222, 339
formatierte Eingabe von 105
lesen von 70, 105, 223
Tastaturpuffer
Zeichen lesen von 609

Tastendruck, Test auf 339
tell (Bibliotheksfunktion) 586
TEMP (Environment-Variable) 598
template
  Dateinamen 384
tempnam (Bibliotheksfunktion) 587
temporäre Dateien
  löschen 462
  Namensvergabe 587, 599
  öffnen 598
  siehe auch Dateien
Terminierungsroutinen 26
Text
  Attribute setzen 588, 590f
  auf Bildschirm ausgeben 434
  Ausschnitt kopieren 389
  in Speicher kopieren 266
  Intensität 360, 393
  Justierung 515
  Parameter für die Ausgabe 517
  Zeichenfarbe 588ff
  Zeichenintensität 305
  Zeile löschen 109
text_info (Datentyp) 267
textattr (Bibliotheksfunktion) 588
Textattribute
  setzen 305, 360, 393
Textausgabe
  BIOS vs. direkter Zugriff 635
  in der Grafik 402, 404
  Parameter 267f
  Platzbedarf 593, 596
  Zeichengröße 520
textbackground (Bibliotheksfunktion) 590
textcolor (Bibliotheksfunktion) 591
Textdateien 101, 104
  lesen 444f, 447
  Modus setzen 509
  siehe auch Dateien
Textfenster
  Cursorposition 280, 626
  Leerzeile einfügen 316
  setzen 627
  Zeile löschen 89, 109
textheight (Bibliotheksfunktion) 593
textmode (Bibliotheksfunktion) 594
Textmodus 398, 434, 537, 627ff
  aktueller 267
  Dateien öffnen im 638
  Koordinaten 266
  setzen 594
  umschalten von Grafik 458
textsettingstype (Datentyp) 269
textwidth (Bibliotheksfunktion) 596
time (Bibliotheksfunktion) 597
timezone (globale Variable) 604, 643
tmpfile (Bibliotheksfunktion) 598
tmpnam (Bibliotheksfunktion) 599
toascii (Bibliotheksfunktion) 600
_tolower (Bibliotheksmakro) 601
tolower (Bibliotheksfunktion) 601
Ton
  erzeugen 539
  ausschalten 393
_toupper (Bibliotheksmakro) 602
toupper (Bibliotheksfunktion) 603
TSR-Programme 71, 340
Turbo Profiler 641
tzname (globale Variable) 604, 643
tzset (Bibliotheksfunktion) 604

U

übergeordnete Prozesse 142, 540
überschreiben
  Dateien 101
Übertragungsparameter 36, 56
Uhrzeit
  daylight (globale Variable) 634
  Differenz zweier Zeitangaben 112
  formatieren für Ausgabe 561
  konvertieren in String 576
  System 106
  siehe auch Datum und Uhrzeit
ultoa (Bibliotheksfunktion) 606
umask (Bibliotheksfunktion) 606
unendliche Werte
  in Gleitkommaoperationen 92
Unendlichkeiten
  und atof 28
ungepufferte Ein-/Ausgaben 522
ungepufferte Streams 489, 521
ungerade Parität 37, 56
ungetc (Bibliotheksfunktion) 608
ungetch (Bibliotheksfunktion) 609
UNIX
  Datum und Uhrzeit konvertieren
    131, 610
unixtodos (Bibliotheksfunktion) 610
unlink (Bibliotheksfunktion) 611
unlock (Bibliotheksfunktion) 612

untergeordnete Prozesse 142, 540
USA
  Datumsformat 97
utime (Bibliotheksfunktion) 613

V
va_arg (Bibliotheksmakro) 615
va_end (Bibliotheksfunktion) 615
va_start (Bibliotheksmakro) 615
variable Parameterzahl 615
Variablen
  Environment 541
  globale 604ff
Vektor-Fonts
  Größe 520
  Installation 320
  Registrierung 454
Vergleich
  zweier Werte 381
Vergleichsroutine, selbstgeschriebene
    63, 363
Verify-Flag 273, 523
_version (globale Variable) 644
Versionsnummern
  DOS 640, 644
Vertauschen der Byte-Reihenfolge 583
Verzeichnis
  absuchen 119, 121, 179, 181, 448
  aktuelles 143, 226, 229
  aktuelles ermitteln 227
  aktuelles wechseln 73
  erzeugen 382
  löschen 461
  Pfad durchsuchen 475f
Verzeichnis-Streams
  lesen aus 448
  öffnen 399
  schließen 87
  zurücksetzen 460
Verzeichnisroutinen
  _dos_findfirst 119
  _dos_findnext 121
  _getdcwd 229
  _makepath 367
  _searchenv 475
  _splitpath 543
  chdir 73
  findfirst 179
  findnext 181
  fnmerge 186
  fnsplit 188
  getcurdir 226
  getcwd 227
  getdisk 232
  mkdir 382
  mktemp 384
  rmdir 461
  rmtmp 462
  searchpath 476
  setdisk 232
Verzögerung, maschinenunabhängige 108
vfprintf (Bibliotheksfunktion) 616
vfscanf (Bibliotheksfunktion) 618
Video-Adapter
  siehe Grafikadapter
Videomodus
  siehe Bildschirmmodus
viewporttype (Datentyp) 274
vollständige Pufferung 522
vprintf (Bibliotheksfunktion) 619
vscanf (Bibliotheksfunktion) 620
vsprintf (Bibliotheksfunktion) 622
vsscanf (Bibliotheksfunktion) 623

W
Währungssymbol 96
Warnton
  ausschalten 393
  erzeugen 539
wchar_t (Array) 625
wcstombs (Bibliotheksfunktion) 625
wctomb (Bibliotheksfunktion) 625
Werte
  absolute 151
  Break- 61, 465
  konvertieren in ASCII-Format 600
  Literale 139
  Potenz berechnen 417
  Speicherbereich füllen mit 508
  vergleichen 372, 381
wherex (Bibliotheksfunktion) 626
wherey (Bibliotheksfunktion) 626
WILDARGS.OBJ (für Jokerzeichen) 5
window (Bibliotheksfunktion) 627
Winkel
  komplexe Zahl 20
Winkeleinheit rad 16, 25
WORD
  aus Schnittstelle lesen 314, 316
  Ausgabe zu I/O-Adresse 401f

Dateiattribut 104
DOS-Attribut 117
lesen aus Speicher 408
Status- 92, 550
wortweise Ausrichtung
Vertauschen von Bytes 583
_write (Bibliotheksfunktion) 628
write (Bibliotheksfunktion) 629
_wscroll (globale Variable) 644

X
X-Koordinate 277
maximale 253f

Y
Y-Koordinate 278

Z
Zahlen
BCD 33, 450
konvertieren von Strings in 27
Rundung 67, 184
Zufalls- 443
Zeichen
Attribute 588, 590f, 634
aus mehreren Bytes 373f, 625
Ausgabe 195, 426f
Farbe setzen 588
Größe 596
hohe Intensität 305
in Strings überschreiben 574
Klassifizierung 329ff
konvertieren in ASCII-Format 600
konvertieren zu Großbuchstaben 602f
konvertieren zu Kleinbuchstaben 601
lesen aus stdin 172, 223
lesen aus Stream 171, 221
lesen von Tastatur 70, 222f
niedrige Intensität 360
normale Intensität 393
suchen in Speicherblöcken 377
suchen in String 553, 571f
Vergrößerungsfaktor 520
Zeilenvorschub 433
Zeichenfarbe
ermitteln 224
für Textausgaben 591
setzen 491
Zeichenfenster
Grenzen 274, 524
löschen 84
setzen 524
zeichenorientierte Geräte 490, 522
Zeichensätze registrieren 454
Zeiger
auf Fehlermeldungen 283, 559
auf lokale Struktur 351
Datei 172, 203ff, 210, 365, 395, 445, 447, 459, 538, 586
far 152, 161f, 383
Frame 359, 503
Stack 359
Zeile
einfügen 316
löschen 89, 109
Zeilenvorschubzeichen 433, 628, 630
zeilenweise Pufferung 522
Zeit
siehe Datum und Uhrzeit
Zeitmessungen
clock 85
Zeitzonen 211, 604
setzen 106
tzname (glob. Variable) 643
Ziffern
prüfen auf 333
Zufallszahlen 439, 443f, 547
Zugriff
lesen/schreiben 208
Zugriffsmodus (Dateien) 75, 77, 606
Datei 124
Zuordnung
Unterdrückung bei scanf 471
Zurückstellen von Zeichen 608
zweidimensionale Balken 30

REFERENZ-
HANDBUCH

BORLAND® C++

BORLAND

BORLAND GMBH, GAUTINGER STR. 10,POSTFACH 1209, D-8130 STARNBERG, TELEFON 0 81 51 / 263-0, TELEFAX 0 81 51 / 142 09
CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001. (408) 438-530
OFFICES IN: AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND, SINGAPORE, SWEDEN AND THE UNITED KINDOM * ARTIKEL-NR. 167303